Introduction to Machine Learning

Types of Learning (Supervised, Unsupervised, Reinforcement, Semi-supervised)

Supervised Learning

1. Conceptual Overview

- **Definition:** Supervised learning is when we train a model using **labeled data** — meaning each input has a corresponding correct output.
- The model learns the mapping from input features X to target output Y.
- Tasks:
    - **Classification:** Predict categorical labels (spam / not spam, disease / no disease).
    - **Regression:** Predict continuous values (house price, temperature).

Real-life example:

- Input: Features of a house (size, location, rooms).
- Output: House price.
- The model learns from past sales (labeled data) to predict prices for new houses.

---

2. Mathematical Foundation

- Training dataset:

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

where $dx_i \in \mathbb{R}^d$ (features), $y_i \in \mathbb{R}$ or categorical (labels).

- Goal: Find a hypothesis (function) h: X → Y such that the prediction error is minimized.
- **Risk (Expected Loss):**

$$R(h) = \mathbb{E}_{(X,Y)\sim P}[L(h(X), Y)]$$

- **Empirical Risk Minimization (ERM):**

  Since the true distribution P is unknown, we minimize empirical risk:

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i)$$

- Common loss functions:
  - Regression → Squared Error: $L = (h(x_i) - y_i)^2$
  - Classification → 0-1 Loss: $L = \mathbf{1}(h(x_i) \neq y_i)$

3. Practical Example (Python)

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

# Load dataset
X, y = load_iris(return_X_y=True)

# Train supervised learning model (classification)
clf = LogisticRegression(max_iter=200)
clf.fit(X, y)

# Predict on a new example
print("Prediction for first sample:", clf.predict([X[0]]))
```

4. Key Takeaways

- **Supervised learning = labeled data.**
- Two main tasks: **Classification** (categorical output), **Regression** (continuous output).
- Uses **loss minimization** to fit the model.
- The foundation for many ML algorithms (Linear Regression, Logistic Regression, Decision Trees, Neural Networks).

Unsupervised Learning

---

1. Conceptual Overview

- **Definition:** Unsupervised learning is when the dataset has **only inputs (features)** but **no labeled outputs**.
- The goal is to uncover **hidden patterns, structure, or relationships** in the data.
- Key tasks:
    - **Clustering:** Group similar data points together (e.g., customer segmentation).
    - **Dimensionality Reduction:** Compress data while preserving structure (e.g., PCA for visualization).
    - **Density Estimation:** Estimate probability distributions from data.
    - **Anomaly Detection:** Identify unusual or rare patterns.

Real-life examples:

- Grouping customers into segments based on purchasing behavior.
- Discovering research paper topics without labels.
- Identifying fraudulent transactions as outliers.

---

2. Mathematical Foundation

- Training dataset:

$$\mathcal{D} = \{x_1, x_2, \ldots, x_n\}, \quad x_i \in \mathbb{R}^d$$

  (no labels y_i).

- **Clustering objective (example: k-means):**

  Assign each point to one of k clusters to minimize within-cluster variance:

$$J = \sum_{i=1}^{n} \min_{j \in \{1,\ldots,k\}} \|x_i - \mu_j\|^2$$

  where μ_j is the centroid of cluster j.

- **Dimensionality Reduction (example: PCA):**

  Find directions (principal components) that maximize variance:

  $\max_w \ \mathrm{Var}(Xw), \quad \|w\| = 1$

3. Practical Example (Python)

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

# Load dataset (iris without labels for unsupervised task)
X, y = load_iris(return_X_y=True)

# Apply k-means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Assign cluster label for first sample
print("Cluster assignment for first sample:", kmeans.predict([X[0]]))
```

4. Key Takeaways

- **Unsupervised learning = unlabeled data.**
- Goal: discover structure (clusters, latent features, anomalies).
- Common techniques: **Clustering (k-means, hierarchical, DBSCAN)**, **Dimensionality Reduction (PCA, t-SNE)**, **Density Estimation**.
- Useful in **exploratory data analysis**, **feature learning**, and **preprocessing** for supervised tasks.

Reinforcement Learning (RL)

---

1. Conceptual Overview

- **Definition:** Reinforcement Learning is a learning paradigm where an **agent** interacts with an **environment** to achieve a goal.
- Instead of being given explicit labels (like in supervised learning), the agent receives **feedback in the form of rewards or penalties** based on its actions.
- Goal: Learn a **policy** (a strategy) that maximizes long-term cumulative reward.

Real-life examples:

- A robot learning to walk.
- An AI playing chess/Go and improving with self-play (AlphaGo).
- A recommendation system improving user engagement.

---

2. Mathematical Foundation

RL is usually modeled as a **Markov Decision Process (MDP)**:

- S: Set of states (agent’s situation).
- A: Set of actions.
- P(s'|s,a): Transition probability (environment dynamics).
- R(s,a): Reward function.
- $\gamma \in [0, 1)$: Discount factor for future rewards.

Objective: Maximize the expected cumulative reward (return):

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Policy (π): A mapping from states to actions.

- Goal is to find the optimal policy π^*:

$$pi^* = \arg\max_{\pi} \; \mathbb{E}[G_t | \pi]$$

Value Function (measures "goodness" of a state or state-action):

- State-value function:

$$V^\pi(s) = \mathbb{E}[G_t | S_t = s, \pi]$$

- Action-value function:

$$Q^\pi(s,a) = \mathbb{E}[G_t | S_t = s, A_t = a, \pi]$$

---

3. Practical Example (Python – simple simulation)

```
import random

# Simple RL environment: 1D world with goal at position 5
state = 0
goal = 5
rewards = 0

for step in range(10):
    action = random.choice([-1, 1])  # agent moves left or right
    state += action
    if state == goal:
        rewards += 10  # reward for reaching goal
        break
    else:
        rewards -= 1  # penalty for each move
```

```
print(f"Final State: {state}, Total Reward: {rewards}")
```

4. Key Takeaways

- RL = **learning by trial and error** using rewards.
- Core components: **Agent, Environment, States, Actions, Rewards**.
- Mathematically modeled using **MDPs**.
- Goal: Find a **policy** that maximizes expected long-term reward.
- Applications: robotics, games (AlphaGo, Atari), recommendation systems, autonomous driving.

Semi-Supervised Learning

1. Conceptual Overview

- **Definition:** Semi-supervised learning uses a **small amount of labeled data** together with a **large amount of unlabeled data**.
- Motivation: Labeling data is often **expensive** (e.g., medical images, legal documents), but unlabeled data is abundant.
- The model leverages structure in the unlabeled data to improve learning from the few labeled samples.

Real-life examples:

- Face recognition: only a few faces are tagged with names, but there are many untagged photos.
- Speech recognition: only some speech samples are transcribed.
- Fraud detection: only a few cases are labeled as fraud, but millions of transactions are unlabeled.

2. Mathematical Foundation

- Dataset:

$$\mathcal{D} = \{(x_1, y_1), \dots, (x_l, y_l)\} \cup \{x_{l+1}, \dots, x_{l+u}\}$$

  where l = labeled samples, u = unlabeled samples, with u $\gg$ l.

- Objective: Improve prediction accuracy by **exploiting both labeled and unlabeled data**.
- Approaches:
  1. **Generative models**: Estimate joint probability p(x,y) using both labeled and unlabeled data.
  2. **Graph-based methods**: Build a similarity graph, propagate labels to nearby unlabeled points.
  3. **Self-training / pseudo-labeling**: Use a model trained on labeled data to predict labels for unlabeled data, then retrain with pseudo-labels.

Mathematically (pseudo-labeling idea):

- Train the initial classifier h_0 on the labeled set.
- Generate pseudo-labels for unlabeled data:

$$\tilde{y}_j = h_0(x_j), \quad j \in \{l+1, \dots, l+u\}$$

- Retrain model on $\{(x_i, y_i)\} \cup \{(x_j, \tilde{y}_j)\}$.

---

3. Practical Example (Python – mock-up with pseudo-labeling)

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris

# Load dataset
X, y = load_iris(return_X_y=True)

# Few labeled samples
labeled_X, labeled_y = X[:10], y[:10]
```

```
unlabeled_X = X[10:]

# Step 1: Train on labeled data
clf = LogisticRegression(max_iter=200)
clf.fit(labeled_X, labeled_y)

# Step 2: Generate pseudo-labels for unlabeled data
pseudo_labels = clf.predict(unlabeled_X)

# Step 3: Combine labeled + pseudo-labeled
X_combined = np.vstack((labeled_X, unlabeled_X))
y_combined = np.hstack((labeled_y, pseudo_labels))

# Retrain
clf.fit(X_combined, y_combined)
print("Semi-supervised training complete")
```

4. Key Takeaways

- Semi-supervised learning = **few labels + many unlabeled samples**.
- Useful when labeling data is expensive or time-consuming.
- Common strategies: **Generative models, graph-based label propagation, pseudo-labeling**.
- Provides a middle ground between **supervised** and **unsupervised** learning.

Machine Learning Methodology

Detailed Conceptual Overview

Machine Learning methodology is the **end-to-end process** of building, evaluating, and improving ML models. Think of it as a **workflow pipeline** that ensures models are not just accurate but also reliable and usable in real-world scenarios.

Step 1: Problem Definition

- Clarify **what kind of problem you are solving**:
  - Classification (predict categories),
  - Regression (predict continuous values),
  - Clustering (grouping data),
  - Recommendation, etc.
- Example: Predicting whether an email is spam → **binary classification**.

👉 Why important? If the problem is not framed correctly, the whole ML pipeline will fail.

Step 2: Data Collection

- Gather **relevant data** from sources (databases, sensors, APIs, user interactions).
- More data usually → better models, but data quality matters more than quantity.
- Example: Collecting housing price data with features like size, location, and number of rooms.

Step 3: Data Preparation & Preprocessing

- Clean the data: handle missing values, remove duplicates, fix inconsistencies.
- Transform the data: scaling, encoding categorical variables, feature engineering.
- Split into **training, validation, and test sets**.
- Example: Standardizing features (so gradient descent converges faster).

👉 This step is often **70–80% of the work** in real ML projects.

Step 4: Model Selection / Hypothesis Choice

- Choose an appropriate model family (hypothesis space):

    - Simple linear models, tree-based models, deep neural networks, etc.
- Choice depends on:
    - Problem type (regression vs classification).
    - Data size and dimensionality.
    - Interpretability vs accuracy trade-offs.

👉 Example: For predicting house prices, start with **Linear Regression** before trying complex models.

Step 5: Training the Model

- Train the model using the training dataset.
- Model parameters (like weights in regression or neural networks) are updated by minimizing a **loss function**.
- Optimization algorithms (e.g., gradient descent) are used here.
- Example: In linear regression, minimize the **sum of squared errors** between predicted and actual values.

Step 6: Validation & Hyperparameter Tuning

- Use a **validation set** to tune hyperparameters (like learning rate, number of trees, and regularization strength).
- Avoid overfitting by checking performance on unseen validation data.
- Example: Choose the best polynomial degree for polynomial regression using validation accuracy.

Step 7: Testing / Evaluation

- Once the best model is chosen, evaluate it on the **test set**, which was kept aside and never seen during training/validation.
- This provides an **unbiased estimate of generalization performance**.
- Example: Report accuracy, precision, recall, and F1-score for a spam detection model.

Step 8: Deployment & Monitoring

- Deploy the model into a real-world system (e.g., web app, mobile app, recommendation engine).
- Monitor model performance over time → data distributions may shift (concept drift).
- Retrain periodically with new data.
- Example: A fraud detection model must adapt as fraud patterns evolve.

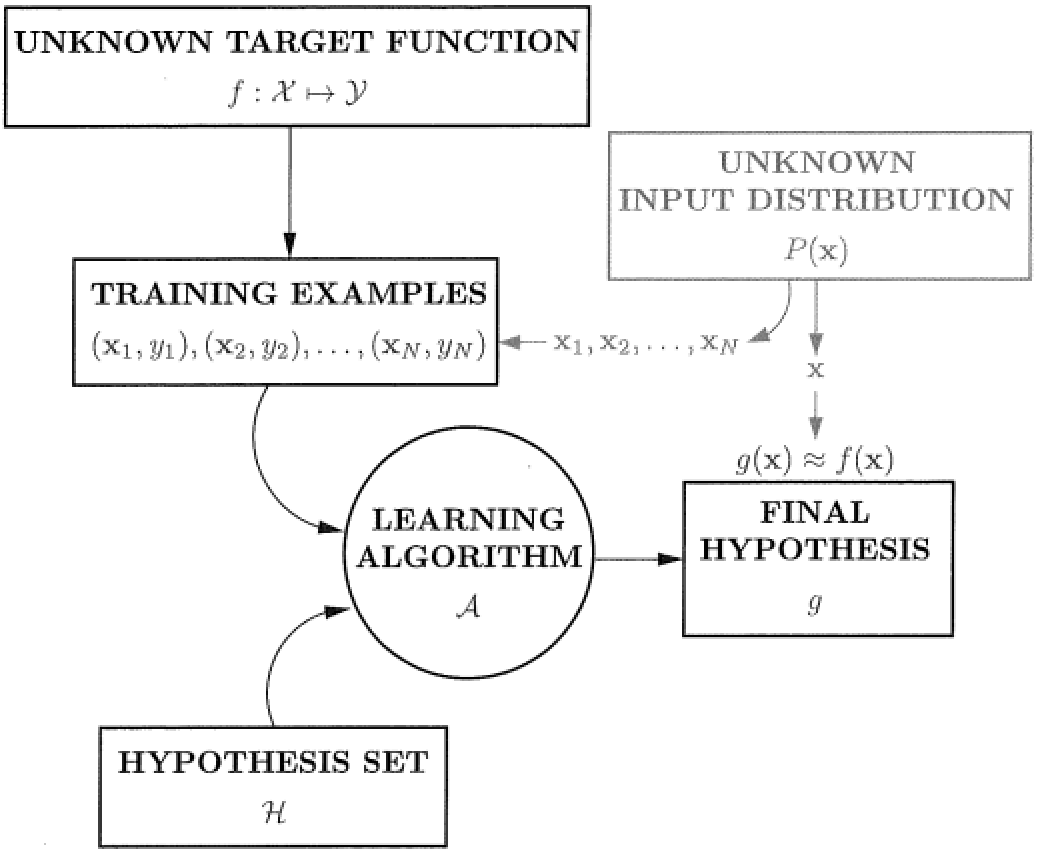
Underlying Principles Throughout

- **Generalization**: The Model must work well on unseen data, not just training data.
- **Bias-Variance Tradeoff**: Balance underfitting (too simple) and overfitting (too complex).
- **Iteration**: ML is rarely linear — you may go back and forth between steps (e.g., collecting more data if accuracy is low).

✅ In short:

ML methodology is about moving from raw data + problem → reliable, deployable, generalizable solution by following a structured pipeline.

✅ Explanation of the Diagram


UNKNOWN TARGET FUNCTION
$f : \mathcal{X} \mapsto \mathcal{Y}$
UNKNOWN INPUT DISTRIBUTION
$P(\mathbf{x})$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
$\mathbf{x}$
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
g
HYPOTHESIS SET
$\mathcal{H}$

1️⃣ UNKNOWN TARGET FUNCTION (f)

- This represents the **real relationship** between the inputs and outputs.
- It's unknown because in practice, we don't know the exact rules or equations that map x (input features) to y (outputs).
- Example: In predicting house prices, f could involve complex factors like location, material, and market trends that we can't fully capture.

2️⃣ UNKNOWN INPUT DISTRIBUTION P(x)

- The way input features are distributed is also unknown.
- We don't know how the data is spread or what patterns it follows in reality.
- We only see sample data points, not the full distribution.

3️⃣ TRAINING EXAMPLES

- We collect sample data $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$ from the real world.
- These are drawn according to P(x) and labeled using f(x), possibly with some noise.
- The training examples are what the learning algorithm uses to figure out the relationship.

4️⃣ HYPOTHESIS SET ($\mathcal{H}$)

- This is the set of possible models or functions that the learning algorithm can choose from.
- It could be all linear functions, polynomials, neural networks, etc.
- The hypothesis set defines the flexibility and constraints of the learning system.

5️⃣ LEARNING ALGORITHM ($\mathcal{A}$)

- The algorithm takes the training examples and searches through the hypothesis set to find the best model.
- It tries to approximate the true function f(x) using the available data and selected model class.

6️⃣ FINAL HYPOTHESIS g

- After learning from the training examples, the algorithm produces a final model g(x).
- This model approximates the true function f(x), but with some error.
- The goal is for g(x) to generalize well to new inputs from the same distribution.

✅ Overall Flow

1. The **unknown target function** f(x) and **input distribution** P(x) generate the training examples.

2. These examples are used by the **learning algorithm**, which searches through the **hypothesis set** $\mathcal{H}$ to find the best model.
3. The algorithm outputs the **final hypothesis** g(x), which we hope closely approximates f(x) for future inputs.

✅ Key Insights

✔️ We never know the true function or data distribution completely — learning is about approximation.

✔️ The choice of hypothesis set and algorithm affects how well the model generalizes.

✔️ Training examples are just samples — good sampling is critical.

✔️ Overfitting and underfitting occur depending on how well the hypothesis set and algorithm handle the data.

✅ Summary

This diagram represents the core structure of machine learning:

- **Real-world uncertainty** → unknown function and distribution
- **Observed data** → training examples
- **Model choices** → hypothesis set and algorithm
- **Goal** → approximate the true relationship as closely as possible

---