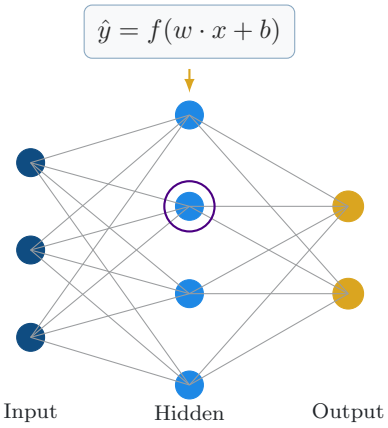
# Machine Learning

## ADP / BS Mathematics

A Comprehensive Course Companion




A journey through concept learning, classification, clustering, probabilistic reasoning, and ensemble methods.

Sohaib Hasan (HOD Mathematics) • Horizon College Chakwal

sohaibhassan199@gmail.com • sohaib-hasan.github.io

META

@plotlab01 /plotlab1

# Contents

# Chapter 1

# Foundations of Machine Learning and Concept Learning

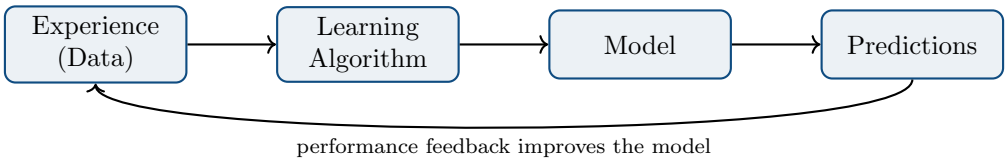
## Introduction to Machine Learning

### What is Machine Learning?

**Machine Learning**

Machine Learning is the field of study that gives computer programs the ability to learn patterns directly from data and improve their performance on a task through experience, rather than being explicitly programmed with a fixed set of rules. Formally, a program is said to **learn** from experience $E$ with respect to a task $T$ and a performance measure $P$, if its performance on $T$, as measured by $P$, improves with experience $E$.

This is similar to fitting a curve through a set of sample points in algebra: the "curve" (model) is not fixed in advance, it is shaped by the data points (experience) themselves.




**Example 1.1 — A Fixed Celsius to Fahrenheit Conversion**

A program always converts temperature using the fixed formula $F = \frac{9}{5}C + 32$, regardless of any data it is given. Is this program an example of Machine Learning?

**Solution:** The program applies the same fixed rule every time and never changes its behaviour based on experience. There is no data-driven improvement in performance.

No, this is traditional programming, not Machine Learning

**Example 1.2 — Predicting House Prices from Past Sales**

A program studies thousands of past house sale records (size, location, price) and uses the patterns found in them to predict the price of a new house; its predictions get more accurate as more sale records are added. Is this program an example of Machine Learning?

**Solution:** The program has an experience $E$ (past sale records), a task $T$ (predicting price), and a performance measure $P$ (prediction accuracy), and $P$ improves as $E$ grows.

Yes, this satisfies the formal definition of Machine Learning

**Example 1.3 — A Spam Filter That Learns from User Feedback**

An email program starts by flagging very few emails as spam. Every time a user manually marks an email as spam or not-spam, the program updates itself, and over time it correctly flags spam more often. Is this program an example of Machine Learning?

**Solution:** The program's experience $E$ is the set of user-marked emails, its task $T$ is classifying new emails, and its performance $P$ (fraction of emails correctly classified) improves as $E$ grows. Yes, this is Machine Learning

**Traditional Programming vs Machine Learning:**

1. **Traditional programming:** a human writes explicit rules; the same input always produces the same output through the same fixed logic (Example 1.1)
2. **Machine Learning:** the program is shown data (experience) and learns the rules or patterns itself; performance improves as more data is seen (Examples 1.2, 1.3)
3. The key test is always the same: is there an experience $E$ that, when increased, improves a measurable performance $P$ on a task $T$?

## Well-Posed Learning Problems

**Well-Posed Learning Problem**

A learning problem is precisely defined, or **well-posed**, once three components are clearly identified:

- **Task $T$:** the specific job the program is learning to perform.
- **Performance measure $P$:** a measurable quantity that tells us how well the program is doing at $T$.
- **Experience $E$:** the data or interaction the program learns from.

**Example 1.4 — Formulating T P E for a Handwriting Recognizer**

A program is being built to recognize hand-written digits (0 to 9) from scanned images, and it is given a large collection of digit images that are already labeled with the correct digit. Identify $T$, $P$, and $E$.

**Solution:**

$T$ : recognizing and classifying hand-written digits in new images

$P$ : percentage of digits correctly classified

$E$ : a collection of digit images with known correct labels

$T, P, E$ identified as above

**Example 1.5 — Formulating T P E for a Checkers-Playing Program**

A program is being built to play the game of checkers, and it improves by playing many practice games against itself and recording which moves led to wins. Identify $T$, $P$, and $E$.

**Solution:**

$T$ : choosing which move to play in a game of checkers
$P$ : percentage of games won against opponents
$E$ : a large number of self-played practice games

$T, P, E$ identified as above

**Example 1.6 — Formulating T P E for a Customer Support Chatbot**

A chatbot is being built to answer customer questions, and it is given a large set of past conversations along with a rating (helpful or not helpful) that customers gave to each chatbot reply. Identify $T$, $P$, and $E$.

**Solution:**

$T$ : generating a reply to a customer's question
$P$ : percentage of replies rated helpful by customers
$E$ : past conversations together with their helpfulness ratings

$T, P, E$ identified as above

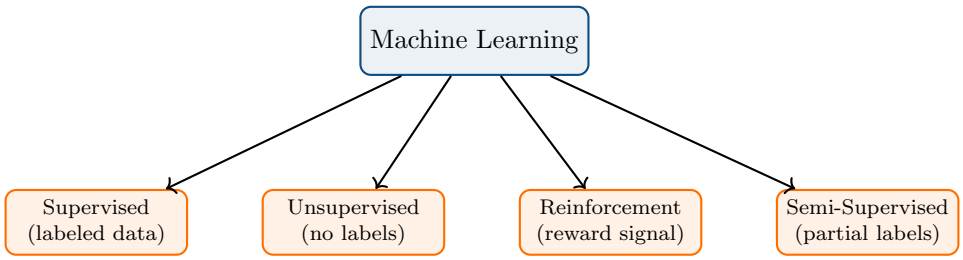
**Quick Checklist for T P E:**

1. $T$ is always an **action or decision** the program performs (classify, predict, choose a move, generate a reply)
2. $P$ is always something you could actually **count or measure** (a percentage, an error rate, a score)
3. $E$ is always the **raw material** the program is trained on (labeled data, self-play games, past interactions)

## Types of Machine Learning

**Types of Machine Learning**

1. **Supervised Learning:** trained on examples that already have the correct output (label) attached; the goal is to predict the output for new, unseen inputs *(labeled data, Chapter 2)*
2. **Unsupervised Learning:** trained on examples with no labels at all; the goal is to discover natural structure or groupings in the data *(no labels, Chapter 3)*
3. **Reinforcement Learning:** an agent learns by interacting with an environment and receiving reward signals for its actions, rather than being shown correct answers directly *(reward-driven, Chapter 4)*
4. **Semi-Supervised Learning:** trained on a small labeled set together with a larger unlabeled set, combining ideas from both supervised and unsupervised learning *(partial labels, Chapter 4)*




**Example 1.7 — Classifying a Learning Scenario (Labeled Emails)**

A dataset contains 500 emails, and every single email is already marked Spam or NotSpam. A program is trained on this dataset to classify future emails. Which type of Machine Learning is this?

**Solution:** Every example has a known label (Spam or NotSpam) attached, and the program is trained directly on these input-output pairs. Supervised Learning

**Example 1.8 — Classifying a Learning Scenario (Unlabeled Purchases)**

A store has 10,000 customer purchase records with no labels of any kind, and wants to discover natural groups of customers with similar buying habits. Which type of Machine Learning is this?

**Solution:** There are no labels anywhere in the data; the goal is purely to discover structure (groups) in the data itself. Unsupervised Learning

**Example 1.9 — Classifying a Learning Scenario (A Game-Playing Robot)**

A robot arm learns to stack blocks by trying different movements; it receives a positive reward when a block is stacked successfully and a negative reward when a block falls, with no example of the "correct" movement ever given directly. Which type of Machine Learning is this?

**Solution:** The robot learns purely through trial-and-error interaction with its environment, guided only by reward and penalty signals, not by labeled examples. Reinforcement Learning

**Distinguishing the Four Types at a Glance:**

1. Ask first: **does every example have a label?** If yes → Supervised. If none do → Unsupervised.
2. Ask: **are only some examples labeled?** If a small labeled portion is combined with a larger unlabeled portion → Semi-Supervised.
3. Ask: **is there an agent taking actions and receiving rewards over time**, rather than a fixed dataset at all? If yes → Reinforcement Learning.
4. These four categories map directly onto Chapters 2 through 4 of these notes, and set up exactly the kind of learning problem studied in the rest of this chapter: Concept Learning (Section 1.2) is a form of supervised learning.

# Concept Learning and General-to-Specific Ordering of Hypotheses

## Concept Learning and Hypothesis Representation

**Concept Learning**

Concept learning is the task of inferring a boolean-valued target function (a **concept**, such as "EnjoySport = Yes") from a set of training examples, each described by attribute values and labeled as a positive or negative instance of the concept. The learner searches through a space of candidate **hypotheses** to find one that best explains the training examples.

**Hypothesis Representation**

A hypothesis $h$ is written as a conjunction (AND) of constraints on each attribute, $h = \langle c_1, c_2, \ldots, c_n \rangle$, where each constraint $c_i$ is one of:

- a **specific value** (e.g. Sunny) — the attribute must equal exactly this value
- **"?"** — any value of this attribute is acceptable
- **"$\varnothing$"** — no value of this attribute is acceptable (this hypothesis rejects every instance)

An instance $x$ **satisfies** $h$ if, for every attribute, $x$'s value matches $h$'s constraint or the constraint is "?".

This is the same idea as a compound logical condition with AND, where each attribute either must match exactly, is left unrestricted, or makes the whole condition impossible to satisfy.

**Example 1.10 — Checking Instance Satisfaction (Specific Hypothesis)**

Let $h = \langle \text{Sunny}, \text{Warm} \rangle$ over attributes $\langle \text{Sky}, \text{AirTemp} \rangle$. Does the instance $x = (\text{Sunny}, \text{Warm})$ satisfy $h$? Does $x = (\text{Rainy}, \text{Warm})$ satisfy $h$?

**Solution:** For $x = (\text{Sunny}, \text{Warm})$: Sky matches (Sunny = Sunny) and AirTemp matches (Warm = Warm). $\boxed{x = (\text{Sunny}, \text{Warm}) \text{ satisfies } h}$ For $x = (\text{Rainy}, \text{Warm})$: Sky does not match (Rainy $\neq$ Sunny). $\boxed{x = (\text{Rainy}, \text{Warm}) \text{ does not satisfy } h}$

**Example 1.11 — Checking Instance Satisfaction (Hypothesis with a "?")**

Let $h = \langle \text{Sunny}, ? \rangle$. Does $x = (\text{Sunny}, \text{Cold})$ satisfy $h$? Does $x = (\text{Rainy}, \text{Cold})$ satisfy $h$?

**Solution:** For $x = (\text{Sunny}, \text{Cold})$: Sky matches (Sunny = Sunny); AirTemp constraint is "?", so any value is accepted. $\boxed{x = (\text{Sunny}, \text{Cold}) \text{ satisfies } h}$ For $x = (\text{Rainy}, \text{Cold})$: Sky does not match (Rainy $\neq$ Sunny), so the "?" on AirTemp does not matter. $\boxed{x = (\text{Rainy}, \text{Cold}) \text{ does not satisfy } h}$

**Example 1.12 — Checking Instance Satisfaction**

Let $h_1 = \langle ?, ? \rangle$ and $h_2 = \langle \varnothing, \varnothing \rangle$. Does $x = (\text{Rainy}, \text{Cold})$ satisfy $h_1$? Does it satisfy $h_2$?

**Solution:** For $h_1 = \langle ?, ? \rangle$: every attribute constraint is "?", so every instance is accepted. $\boxed{x \text{ satisfies } h_1 \text{ (in fact every possible instance satisfies } h_1)}$ For $h_2 = \langle \varnothing, \varnothing \rangle$: the constraint $\varnothing$ rejects every value. $\boxed{x \text{ does not satisfy } h_2 \text{ (no instance ever satisfies } h_2)}$

**Reading Hypothesis Constraints:**

1. A **specific value** constraint accepts only instances matching that exact value
2. **"?"** accepts every value of that attribute — the least restrictive constraint
3. **"∅"** accepts no value of that attribute — the most restrictive constraint, making the whole hypothesis reject everything
4. $h = \langle ?, ?, \ldots, ? \rangle$ is the **most general** hypothesis (accepts every instance); $h = \langle \varnothing, \varnothing, \ldots, \varnothing \rangle$ is the **most specific** hypothesis (accepts no instance)

## General-to-Specific Ordering of Hypotheses

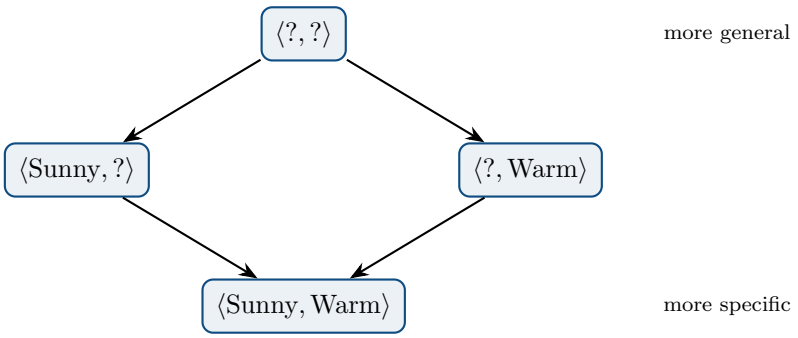
### More-General-Than-Or-Equal-To Relation

For two hypotheses $h_j$ and $h_k$ defined over the same attributes, $h_j$ is more-general-than-or-equal-to $h_k$, written $h_j \geq_g h_k$, if every instance that satisfies $h_k$ also satisfies $h_j$. If in addition some instance satisfies $h_j$ but not $h_k$, then $h_j$ is **strictly** more general than $h_k$, written $h_j >_g h_k$.

This is the same idea as a subset relation: $h_j \geq_g h_k$ means the set of instances accepted by $h_k$ is a subset of the set of instances accepted by $h_j$.

### Properties of the General-to-Specific Ordering

1. The relation $\geq_g$ is a **partial order**: not every pair of hypotheses is comparable *(some pairs are incomparable)*
2. $\langle ?, ?, \ldots, ? \rangle$ is more general than (or equal to) every hypothesis in $H$ *(top of the ordering)*
3. $\langle \varnothing, \varnothing, \ldots, \varnothing \rangle$ is more specific than (or equal to) every hypothesis in $H$ *(bottom of the ordering)*
4. Replacing a "?" with a specific value, or a specific value with "∅", always produces a **more specific** hypothesis *(specialization step)*




### Example 1.13 — Comparing a General and a Specific Hypothesis

Let $h_1 = \langle \text{Sunny}, ? \rangle$ and $h_2 = \langle \text{Sunny}, \text{Warm} \rangle$, over Sky $\in \{\text{Sunny}, \text{Rainy}\}$ and AirTemp $\in \{\text{Warm}, \text{Cold}\}$. Determine whether $h_1 \geq_g h_2$.

**Solution:** List instances satisfying each hypothesis.

$$\text{Instances satisfying } h_2 = \{(\text{Sunny}, \text{Warm})\}$$
$$\text{Instances satisfying } h_1 = \{(\text{Sunny}, \text{Warm}),\ (\text{Sunny}, \text{Cold})\}$$

Every instance satisfying $h_2$ also satisfies $h_1$. $\boxed{h_1 \geq_g h_2 \quad (\text{in fact } h_1 >_g h_2)}$

**Example 1.14 — Comparing a Hypothesis with "?" in a Different Position**

Let $h_1 = \langle ?, \text{Warm} \rangle$ and $h_2 = \langle \text{Sunny}, \text{Warm} \rangle$. Determine whether $h_1 \geq_g h_2$.

**Solution:**

$$\text{Instances satisfying } h_2 = \{(\text{Sunny}, \text{Warm})\}$$
$$\text{Instances satisfying } h_1 = \{(\text{Sunny}, \text{Warm}),\ (\text{Rainy}, \text{Warm})\}$$

Every instance satisfying $h_2$ also satisfies $h_1$. $\boxed{h_1 \geq_g h_2 \quad (\text{in fact } h_1 >_g h_2)}$

**Example 1.15 — Two Incomparable Hypotheses**

Let $h_1 = \langle \text{Sunny}, ? \rangle$ and $h_2 = \langle ?, \text{Warm} \rangle$. Determine whether $h_1 \geq_g h_2$ or $h_2 \geq_g h_1$.

**Solution:**

$$\text{Instances satisfying } h_1 = \{(\text{Sunny}, \text{Warm}),\ (\text{Sunny}, \text{Cold})\}$$
$$\text{Instances satisfying } h_2 = \{(\text{Sunny}, \text{Warm}),\ (\text{Rainy}, \text{Warm})\}$$

Neither set contains the other: $(\text{Sunny}, \text{Cold})$ satisfies $h_1$ but not $h_2$, and $(\text{Rainy}, \text{Warm})$ satisfies $h_2$ but not $h_1$. $\boxed{h_1 \text{ and } h_2 \text{ are incomparable — neither is more general than the other}}$ This confirms Property 1 of the theorembox above: the ordering is only a *partial* order, since not every pair of hypotheses can be compared. ✓

**How to Compare Two Hypotheses:**

1. List (or imagine) the instances each hypothesis accepts
2. If every instance accepted by $h_2$ is also accepted by $h_1$, then $h_1 \geq_g h_2$
3. If neither hypothesis's accepted set contains the other, the two hypotheses are **incomparable**
4. This ordering is exactly what guides a search through hypothesis space: moving up means generalizing, moving down means specializing

## The Find-S Algorithm

**Find-S**

Find-S is a concept learning algorithm that searches the hypothesis space and outputs the **most specific** hypothesis that is consistent with all the **positive** training examples. It starts from the most specific hypothesis $\langle \varnothing, \ldots, \varnothing \rangle$ and generalizes it, one attribute at a time, only when a positive example is not yet satisfied. Negative examples are completely ignored.

**Find-S Algorithm**

**Input:** a set of training examples, each labeled positive or negative
**Output:** the most specific hypothesis $h$ consistent with all positive examples

1. Initialize $h$ to the most specific hypothesis in $H$: $h = \langle \varnothing, \varnothing, \ldots, \varnothing \rangle$.
2. For each training example $x$: if $x$ is negative, ignore it and move to the next example.
3. If $x$ is positive, check every attribute constraint in $h$ against $x$.
4. For each constraint not satisfied by $x$: if the constraint is $\varnothing$, replace it with $x$'s value for that attribute; if it is a specific value different from $x$'s value, replace it with "?".
5. Repeat steps 2 to 4 for every remaining training example.
6. Output the final hypothesis $h$.

**Example 1.16 — Tracing Find-S on a Weather Dataset**

| Sky | AirTemp | EnjoySport |
|---|---|---|
| Sunny | Warm | Yes |
| Sunny | Warm | Yes |
| Rainy | Cold | No |
| Sunny | Cold | Yes |

Trace Find-S on this dataset and give the final hypothesis.

**Solution:**

$$h_0 = \langle \varnothing, \varnothing \rangle \quad \text{(initial, most specific)}$$

**Example 1 (Sunny, Warm, Yes) — positive:** both constraints are $\varnothing$, so update each to match.

$$h_1 = \langle \text{Sunny}, \text{Warm} \rangle$$

**Example 2 (Sunny, Warm, Yes) — positive:** already matches $h_1$, no change.

$$h_2 = \langle \text{Sunny}, \text{Warm} \rangle$$

**Example 3 (Rainy, Cold, No) — negative:** Find-S ignores negative examples.

$$h_3 = h_2 = \langle \text{Sunny}, \text{Warm} \rangle$$

**Example 4 (Sunny, Cold, Yes) — positive:** Sky matches (Sunny), but AirTemp does not (Warm $\neq$ Cold), so generalize AirTemp to "?".

$$h_4 = \langle \text{Sunny}, ? \rangle$$

Final hypothesis: $h = \langle \text{Sunny}, ? \rangle$

**Example 1.17 — Tracing Find-S on a Fruit Ripeness Dataset**

| Color | Size | Ripe |
|---|---|---|
| Red | Big | Yes |
| Red | Big | Yes |
| Green | Small | No |
| Red | Small | Yes |

Trace Find-S on this dataset and give the final hypothesis.

**Solution:**

$$h_0 = \langle \varnothing, \varnothing \rangle$$

**Example 1 (Red, Big, Yes) — positive:** update both constraints from $\varnothing$ to match.

$$h_1 = \langle \text{Red}, \text{Big} \rangle$$

**Example 2 (Red, Big, Yes) — positive:** already matches, no change.

$$h_2 = \langle \text{Red}, \text{Big} \rangle$$

**Example 3 (Green, Small, No) — negative:** ignored.

$$h_3 = h_2 = \langle \text{Red}, \text{Big} \rangle$$

**Example 4 (Red, Small, Yes) — positive:** Color matches (Red), but Size does not (Big $\neq$ Small), so generalize Size to "?".

$$h_4 = \langle \text{Red}, ? \rangle$$

Final hypothesis: $h = \langle \text{Red}, ? \rangle$

**Example 1.18 — Tracing Find-S to the Most General Hypothesis**

| Shape | Color | Target |
|---|---|---|
| Round | Red | Yes |
| Round | Green | Yes |
| Square | Red | Yes |

Trace Find-S on this dataset and give the final hypothesis.

**Solution:**

$$h_0 = \langle \varnothing, \varnothing \rangle$$

**Example 1 (Round, Red, Yes) — positive:** update both constraints from $\varnothing$ to match.

$$h_1 = \langle \text{Round}, \text{Red} \rangle$$

**Example 2 (Round, Green, Yes) — positive:** Shape matches (Round), but Color does not (Red $\neq$ Green), so generalize Color to "?".

$$h_2 = \langle \text{Round}, ? \rangle$$

**Example 3 (Square, Red, Yes) — positive:** Shape does not match (Round $\neq$ Square), so generalize Shape to "?"; Color is already "?", so it stays.

$$h_3 = \langle ?, ? \rangle$$

Final hypothesis: $h = \langle ?, ? \rangle$ Since every positive example forced a generalization, Find-S was pushed all the way to the most general hypothesis — meaning no attribute in this small dataset was restrictive enough to separate positive from negative examples. ✓

**Limitations of Find-S:**

1. Find-S **ignores negative examples entirely**, so it cannot detect if the training data is inconsistent (e.g. two identical instances labeled differently)
2. Find-S outputs only **one** hypothesis (the most specific one consistent with the positives); it cannot tell us whether this was the *only* consistent hypothesis or one of many
3. Find-S has no way to know if it has converged to the true target concept, since it never checks negative examples for consistency
4. These limitations are exactly what motivate the **Candidate Elimination Algorithm** (Section 1.3), which tracks *all* hypotheses consistent with *both* positive and negative examples at once

# Version Spaces and the Candidate Elimination Algorithm

## Version Spaces

**Version Space**

Given a hypothesis space $H$ and a set of training examples $D$, the version space $VS_{H,D}$ is the subset of all hypotheses in $H$ that are consistent with every training example in $D$ — that is, every hypothesis that correctly classifies every positive example as positive and every negative example as negative.

Find-S (Section 1.2.3) returns only **one** member of this set (the most specific one); the version space is the **entire** set of hypotheses that fit the data equally well.

**Specific and General Boundary Sets**

Rather than listing every hypothesis in the (possibly huge) version space one by one, it is represented compactly using two boundary sets:

- **Specific boundary $S$:** the set of most specific hypotheses in the version space.
- **General boundary $G$:** the set of most general hypotheses in the version space.

Every hypothesis $h$ satisfying $S \leq_g h \leq_g G$ for some member of each boundary is guaranteed to be in the version space, without needing to be listed individually.

## The Candidate Elimination Algorithm

**Candidate Elimination Algorithm**

**Input:** a set of training examples, each labeled positive or negative
**Output:** the boundary sets $S$ and $G$ describing the final version space

1. Initialize $S$ to the most specific hypothesis in $H$: $S = \{\langle \varnothing, \ldots, \varnothing \rangle\}$.
2. Initialize $G$ to the most general hypothesis in $H$: $G = \{\langle ?, \ldots, ? \rangle\}$.
3. For each training example $d$:

   **If $d$ is positive:**

   (a) Remove from $G$ any hypothesis inconsistent with $d$ (any hypothesis that rejects $d$).

   (b) For each hypothesis $s$ in $S$ inconsistent with $d$: remove $s$, and replace it with all of its minimal generalizations that are consistent with $d$ and are more specific than some hypothesis in $G$.

   (c) Remove from $S$ any hypothesis that is more general than another hypothesis in $S$.

   **If $d$ is negative:**

   (a) Remove from $S$ any hypothesis inconsistent with $d$ (any hypothesis that accepts $d$).

   (b) For each hypothesis $g$ in $G$ inconsistent with $d$: remove $g$, and replace it with all of its minimal specializations that are consistent with $d$ and are more general than some hypothesis in $S$.

(c) Remove from $G$ any hypothesis that is more specific than another hypothesis in $G$.

4. Repeat step 3 for every remaining training example.
5. Return the final $S$ and $G$.

**Example 1.19 — Full Trace Converging to a Single Hypothesis**

| # | Sky | AirTemp | EnjoySport |
|---|---|---|---|
| 1 | Sunny | Warm | Yes |
| 2 | Sunny | Cold | No |
| 3 | Rainy | Warm | Yes |

Trace the Candidate Elimination Algorithm on this dataset.

**Solution:**

$$S_0 = \{\langle \varnothing, \varnothing \rangle\}, \qquad G_0 = \{\langle ?, ? \rangle\}$$

**Example 1 (Sunny, Warm, Yes) — positive:** generalize $S_0$ minimally to cover it.

$$S_1 = \{\langle \text{Sunny}, \text{Warm} \rangle\}, \qquad G_1 = \{\langle ?, ? \rangle\} \text{ (already accepts it, unchanged)}$$

**Example 2 (Sunny, Cold, No) — negative:** $S_1$ already rejects it (unchanged). Specialize $G_1$ minimally to exclude it, keeping every result more general than $S_1$.

$$\text{Try Sky} \to \text{Rainy} : \langle \text{Rainy}, ? \rangle \text{ —rejects positive example 1, discard}$$
$$\text{Try AirTemp} \to \text{Warm} : \langle ?, \text{Warm} \rangle \text{ —accepts positive example 1, keep}$$

$$S_2 = \{\langle \text{Sunny}, \text{Warm} \rangle\}, \qquad G_2 = \{\langle ?, \text{Warm} \rangle\}$$

**Example 3 (Rainy, Warm, Yes) — positive:** $S_2$ does not accept it (Sky mismatch); generalize minimally.

$$S_3 = \{\langle ?, \text{Warm} \rangle\}$$

$G_2$ already accepts (Rainy, Warm) (AirTemp matches Warm), so it is unchanged: $G_3 = \{\langle ?, \text{Warm} \rangle\}$. $\boxed{S_3 = G_3 = \{\langle ?, \text{Warm} \rangle\}}$ Since $S$ and $G$ have converged to a single, identical hypothesis, the version space contains exactly one hypothesis — the target concept has been fully learned.

**Example 1.20 — A Trace with a Branching General Boundary**

| # | Sky | Wind | Target |
|---|---|---|---|
| 1 | Sunny | Weak | Yes |
| 2 | Rainy | Strong | No |
| 3 | Sunny | Strong | Yes |

Trace the Candidate Elimination Algorithm on this dataset.

**Solution:**

$$S_0 = \{\langle \varnothing, \varnothing \rangle\}, \qquad G_0 = \{\langle ?, ? \rangle\}$$

**Example 1 (Sunny, Weak, Yes) — positive:**

$$S_1 = \{\langle \text{Sunny}, \text{Weak} \rangle\}, \qquad G_1 = \{\langle ?, ? \rangle\}$$

**Example 2 (Rainy, Strong, No) — negative:** specialize $G_1$ minimally; both single-attribute specializations remain consistent with the positive example seen so far.

$$\text{Sky} \to \text{Sunny} : \langle \text{Sunny}, ? \rangle \text{ —accepts example 1, keep}$$
$$\text{Wind} \to \text{Weak} : \langle ?, \text{Weak} \rangle \text{ —accepts example 1, keep}$$

$$S_2 = \{\langle \text{Sunny}, \text{Weak} \rangle\}, \qquad G_2 = \{\langle \text{Sunny}, ? \rangle,\ \langle ?, \text{Weak} \rangle\}$$

**Example 3 (Sunny, Strong, Yes) — positive:** $S_2$ does not accept it (Wind mismatch); generalize minimally.

$$S_3 = \{\langle \text{Sunny}, ? \rangle\}$$

Check each member of $G_2$ against this positive example:

$$\langle \text{Sunny}, ? \rangle : \text{accepts (Sunny,Strong)} \Rightarrow \text{keep}$$
$$\langle ?, \text{Weak} \rangle : \text{Wind=Weak} \neq \text{Strong} \Rightarrow \text{rejects positive} \Rightarrow \text{discard}$$

$$G_3 = \{\langle \text{Sunny}, ? \rangle\}$$

$\boxed{S_3 = G_3 = \{\langle \text{Sunny}, ? \rangle\}}$ The negative example first caused $G$ to **branch** into two candidate hypotheses; the next positive example then eliminated the branch inconsistent with it, and $S$ and $G$ converged together.

**Example 1.21 — Enumerating the Full Version Space from Boundaries**

| # | Taste | Size | Target |
|---|---|---|---|
| 1 | Sweet | Big | No |
| 2 | Sour | Small | Yes |

Trace the Candidate Elimination Algorithm and list every hypothesis in the final version space.

**Solution:**

$$S_0 = \{\langle \varnothing, \varnothing \rangle\}, \qquad G_0 = \{\langle ?, ? \rangle\}$$

**Example 1 (Sweet, Big, No) — negative:** $S_0$ already rejects everything (unchanged). Specialize $G_0$ minimally.

$$\text{Taste} \to \text{Sour} : \langle \text{Sour}, ? \rangle$$
$$\text{Size} \to \text{Small} : \langle ?, \text{Small} \rangle$$

(No positive example has been seen yet, so both specializations are valid.)

$$S_1 = \{\langle \varnothing, \varnothing \rangle\}, \qquad G_1 = \{\langle \text{Sour}, ? \rangle,\ \langle ?, \text{Small} \rangle\}$$

**Example 2 (Sour, Small, Yes) — positive:** generalize $S_1$ minimally to cover it.

$$S_2 = \{\langle \text{Sour}, \text{Small} \rangle\}$$

Check each member of $G_1$: both accept (Sour,Small), so both are kept.

$$G_2 = \{\langle \text{Sour}, ? \rangle,\ \langle ?, \text{Small} \rangle\}$$

$\boxed{\text{Final: } S = \{\langle \text{Sour}, \text{Small} \rangle\}, \quad G = \{\langle \text{Sour}, ? \rangle,\ \langle ?, \text{Small} \rangle\}}$ Every hypothesis $h$ between $S$ and some member of $G$ belongs to the version space:

$$\text{Version Space} = \{\ \langle \text{Sour}, \text{Small} \rangle,\ \ \langle \text{Sour}, ? \rangle,\ \ \langle ?, \text{Small} \rangle\ \}$$

**Reading the Boundary Sets:**

1. $S$ always generalizes on **positive** examples only; $G$ always specializes on **negative** examples only
2. If $S = G$ and both contain exactly one hypothesis, the target concept has been **uniquely** identified (Examples 1.19, 1.20)
3. If $S$ and $G$ have not converged, **every** hypothesis lying between them (more general than $S$, more specific than $G$) is still a valid candidate, as enumerated in Example 1.21
4. Unlike Find-S, Candidate Elimination uses **both** positive and negative examples, so it can detect an inconsistent training set: this happens if $S$ or $G$ ever becomes empty

## Practice Problems

1. A program is built to recommend movies to users, and it improves its recommendations by studying which past recommendations users actually watched versus skipped. Identify $T$, $P$, and $E$ for this learning problem.

2. A program is built to detect fraudulent credit card transactions, trained on a large history of past transactions each labeled Fraud or NotFraud. Identify $T$, $P$, and $E$ for this learning problem.

3. A fitness app groups 5,000 users into activity patterns using only their step-count and heart-rate data, with no labels of any kind provided. Which type of Machine Learning is this, and why?

4. A self-driving car's obstacle-avoidance module learns by taking actions in a simulated environment and receiving a penalty every time it comes too close to an obstacle. Which type of Machine Learning is this, and why?

5. Let $h = \langle \text{Rainy}, \text{Cold} \rangle$ over $\text{Sky} \in \{\text{Sunny}, \text{Rainy}\}$ and $\text{AirTemp} \in \{\text{Warm}, \text{Cold}\}$. Does the instance $x = (\text{Rainy}, \text{Cold})$ satisfy $h$? Does $x = (\text{Sunny}, \text{Cold})$ satisfy $h$?

6. Let $h_1 = \langle \text{Rainy}, ? \rangle$ and $h_2 = \langle ?, \text{Cold} \rangle$. Determine whether $h_1 \geq_g h_2$, $h_2 \geq_g h_1$, or the two are incomparable.

7. Let $h_1 = \langle ?, \text{Cold} \rangle$ and $h_2 = \langle \text{Rainy}, \text{Cold} \rangle$. Determine whether $h_1 \geq_g h_2$, and state whether the relation is strict.

8. Trace the Find-S algorithm on the dataset below and give the final hypothesis.

| Color | Shape | Target |
|---|---|---|
| Red | Round | Yes |
| Green | Round | No |
| Red | Square | Yes |
| Red | Round | Yes |

9. Trace the Find-S algorithm on the dataset below and give the final hypothesis.

| Texture | Temperature | Target |
|---|---|---|
| Smooth | Hot | Yes |
| Smooth | Cold | Yes |
| Rough | Hot | No |

10. In one or two lines, explain why the version space can be represented using only the boundary sets $S$ and $G$ instead of listing every consistent hypothesis individually.

11. Trace the Candidate Elimination Algorithm on the dataset below, showing $S$ and $G$ after every example, and give the final version space.

| # | Size | Color | Target |
|---|---|---|---|
| 1 | Big | Red | Yes |
| 2 | Small | Red | No |
| 3 | Big | Blue | Yes |

12. Trace the Candidate Elimination Algorithm on the dataset below, showing $S$ and $G$ after every example, and give the final version space.

| # | Humidity | Wind | Target |
|---|---|---|---|
| 1 | High | Weak | No |
| 2 | Normal | Weak | Yes |
| 3 | Normal | Strong | Yes |

13. After processing some training examples, a learner has $S = \{\langle \text{Red}, ? \rangle\}$ and $G = \{\langle \text{Red}, ? \rangle,\ \langle ?, \text{Big} \rangle\}$. List every hypothesis contained in this version space.

14. Given $S = \{\langle \text{Sunny}, \text{Warm} \rangle\}$ and $G = \{\langle \text{Sunny}, ? \rangle,\ \langle ?, \text{Warm} \rangle\}$, a new negative example $(\text{Sunny}, \text{Cold})$ is observed. Update $S$ and $G$ accordingly.

15. In two or three lines, explain one key advantage the Candidate Elimination Algorithm has over Find-S, and one situation in which their final specific-boundary hypothesis $S$ would come out identical.

# Chapter 2

# Supervised Learning: Classification and Regression

## Decision Trees

### Introduction to Decision Trees

**Decision Tree**

A decision tree is a tree-shaped classification model in which every internal node tests one attribute of the input, every branch represents the outcome of that test, and every leaf node assigns a class label. To classify a new example, start at the root and follow the branch matching the example's attribute values, node after node, until a leaf is reached.

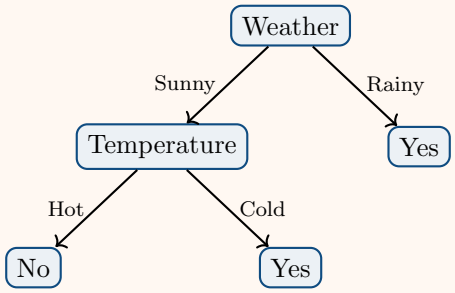
This is nothing more than a nested sequence of if-else questions applied to the attributes of an example.

**Example 2.1 — Classifying an Instance from a Cricket Tree**

The table below shows whether Ali played cricket based on the weather.

| Weather | Temperature | PlayCricket |
|---|---|---|
| Sunny | Hot | No |
| Sunny | Cold | Yes |
| Rainy | Hot | Yes |
| Rainy | Cold | Yes |

The decision tree built from this data is shown below.




Use this tree to classify a new instance: Weather = Sunny Temperature = Cold.

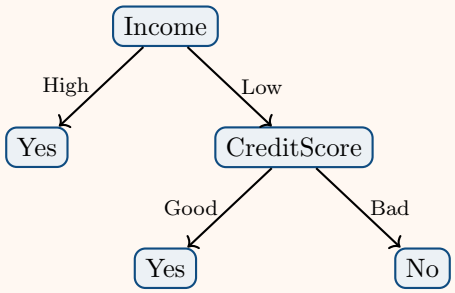
**Solution:** Start at the root. Weather = Sunny, so move to the Temperature node. Temperature = Cold, so move to the right branch, which is a leaf labeled Yes.
Predicted class: Yes

**Example 2.2 — Classifying a New Loan Applicant**

The table below shows loan approval decisions.

| Income | CreditScore | LoanApproved |
|---|---|---|
| High | Good | Yes |
| High | Bad | Yes |
| Low | Good | Yes |
| Low | Bad | No |

The decision tree built from this data is shown below.




Use this tree to classify a new applicant: Income = Low CreditScore = Good.

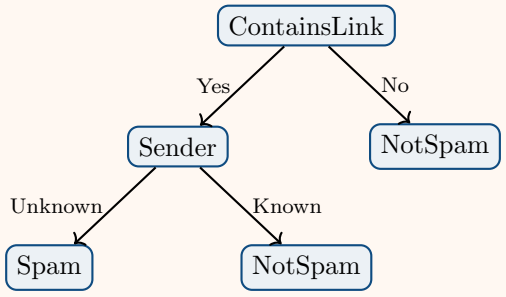
**Solution:** Start at the root. Income = Low, so move to the CreditScore node. CreditScore = Good, so move to the left branch, which is a leaf labeled Yes.
Predicted class: Yes (Approved)

**Example 2.3 — Classifying a New Email**

The table below shows whether emails are spam.

| ContainsLink | Sender | Class |
|---|---|---|
| Yes | Unknown | Spam |
| Yes | Known | NotSpam |
| No | Unknown | NotSpam |
| No | Known | NotSpam |

The decision tree built from this data is shown below.




Use this tree to classify a new email: ContainsLink = Yes Sender = Known.

**Solution:** Start at the root. ContainsLink = Yes, so move to the Sender node. Sender = Known, so move to the right branch, which is a leaf labeled NotSpam.
Predicted class: NotSpam

**Decision Tree Terminology:**

1. **Root node** the topmost node where classification always begins
2. **Internal node** a node that tests one attribute and branches on its values
3. **Branch (edge)** represents one possible value of the tested attribute
4. **Leaf node** a terminal node that outputs a class label with no further test

## Entropy and Impurity

**Entropy**

For a set $S$ containing examples from $c$ classes, entropy measures how mixed or impure the set is:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

where $p_i$ is the proportion of examples in $S$ belonging to class $i$.

This is the same weighted-average-of-surprise idea used in information theory.

**Properties of Entropy**

1. $H(S) = 0$ when every example in $S$ belongs to the same class *(pure set)*
2. $H(S)$ is largest when the classes are split as evenly as possible *(maximum impurity)*
3. $0 \leq H(S) \leq \log_2 c$ for a set with $c$ classes *(bounds)*

**Example 2.4 — Entropy of a Mixed Exam Result Set**

A dataset $S$ of 6 students contains 4 examples labeled Pass and 2 examples labeled Fail. Compute $H(S)$.

**Solution:**

$$p(\text{Pass}) = \frac{4}{6} = \frac{2}{3}, \qquad p(\text{Fail}) = \frac{2}{6} = \frac{1}{3}$$

$$\begin{aligned} H(S) &= -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} \\ &= -\frac{2}{3}(-0.585) - \frac{1}{3}(-1.585) \\ &= 0.390 + 0.528 \end{aligned}$$

$$\boxed{H(S) \approx 0.918}$$

**Example 2.5 — Entropy of a Pure Fruit Set**

A dataset $S$ of 5 fruits contains 5 examples all labeled Mango. Compute $H(S)$.

**Solution:**

$$p(\text{Mango}) = \frac{5}{5} = 1$$

$$H(S) = -(1)\log_2(1) = -(1)(0)$$

$\boxed{H(S) = 0}$ Since every example belongs to one class, the set is completely pure, matching Property 1 above. ✓

**Example 2.6 — Entropy of a Balanced Coin Toss Set**

A dataset $S$ of 6 coin tosses contains 3 examples labeled Heads and 3 examples labeled Tails. Compute $H(S)$.

**Solution:**

$$p(\text{Heads}) = \frac{3}{6} = \frac{1}{2}, \qquad p(\text{Tails}) = \frac{3}{6} = \frac{1}{2}$$

$$\begin{aligned} H(S) &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} \\ &= -\frac{1}{2}(-1) - \frac{1}{2}(-1) \\ &= 0.5 + 0.5 \end{aligned}$$

$\boxed{H(S) = 1}$ This is the largest possible entropy for a two-class set, matching Property 2 above. ✓

**Quick Reference:**

1. $\log_2 2 = 1$ and $\log_2 3 \approx 1.585$ (values needed for almost every entropy calculation in this chapter)
2. $H(S) = 0$ means pure and $H(S) = 1$ means maximum impurity for a two-class set
3. Lower entropy is always better for building compact decision trees

## Information Gain

**Information Gain**

Information Gain measures how much entropy is reduced by splitting a set $S$ on attribute $A$:

$$\text{Gain}(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

where $S_v$ is the subset of $S$ for which attribute $A$ has value $v$.

Think of it as the total entropy minus a weighted average of the entropies of the resulting pieces.

**Example 2.7 — Information Gain of Weather**

Consider the dataset below.

| Weather | Temperature | PlayCricket |
|---|---|---|
| Sunny | Hot | No |
| Sunny | Cold | No |
| Sunny | Hot | Yes |
| Rainy | Cold | Yes |
| Rainy | Cold | Yes |
| Rainy | Hot | Yes |

Compute $\text{Gain}(S, \text{Weather})$.

**Solution:** The set has 4 Yes and 2 No, the same proportion as Example 2.4, so $H(S) \approx 0.918$.

$$S_{\text{Sunny}} = \{\text{No,No,Yes}\}: \ 1 \text{ Yes}, 2 \text{ No} \Rightarrow H(S_{\text{Sunny}}) \approx 0.918$$
$$S_{\text{Rainy}} = \{\text{Yes,Yes,Yes}\}: \ \text{pure} \Rightarrow H(S_{\text{Rainy}}) = 0$$
$$\text{Weighted entropy} = \frac{3}{6}(0.918) + \frac{3}{6}(0) = 0.459$$
$$\text{Gain}(S, \text{Weather}) = 0.918 - 0.459$$

$\boxed{\text{Gain}(S, \text{Weather}) \approx 0.459}$

**Example 2.8 — Information Gain of Temperature**

Using the same dataset as Example 2.7, compute $\text{Gain}(S, \text{Temperature})$.

**Solution:**

$$S_{\text{Hot}} = \{\text{No,Yes,Yes}\}: \ 2 \text{ Yes}, 1 \text{ No} \Rightarrow H(S_{\text{Hot}}) \approx 0.918$$
$$S_{\text{Cold}} = \{\text{No,Yes,Yes}\}: \ 2 \text{ Yes}, 1 \text{ No} \Rightarrow H(S_{\text{Cold}}) \approx 0.918$$
$$\text{Weighted entropy} = \frac{3}{6}(0.918) + \frac{3}{6}(0.918) = 0.918$$
$$\text{Gain}(S, \text{Temperature}) = 0.918 - 0.918$$

$\boxed{\text{Gain}(S, \text{Temperature}) = 0}$ Since $\text{Gain(Weather)} \approx 0.459 > \text{Gain(Temperature)} = 0$, Weather is the better attribute to split on first.

**Example 2.9 — Information Gain of Sky Condition**

Consider a dataset about carrying an umbrella.

| Sky | Humidity | TakeUmbrella |
|---|---|---|
| Cloudy | High | Yes |
| Cloudy | Low | Yes |
| Cloudy | High | Yes |
| Clear | Low | No |
| Clear | Low | No |
| Clear | High | Yes |

Compute $\text{Gain}(S, \text{Sky})$.

**Solution:** The set has 4 Yes and 2 No, so $H(S) \approx 0.918$.

$$S_{\text{Cloudy}} = \{\text{Yes,Yes,Yes}\}: \ \text{pure} \Rightarrow H(S_{\text{Cloudy}}) = 0$$
$$S_{\text{Clear}} = \{\text{No,No,Yes}\}: \ 1 \text{ Yes}, 2 \text{ No} \Rightarrow H(S_{\text{Clear}}) \approx 0.918$$
$$\text{Weighted entropy} = \frac{3}{6}(0) + \frac{3}{6}(0.918) = 0.459$$
$$\text{Gain}(S, \text{Sky}) = 0.918 - 0.459$$

$\boxed{\text{Gain}(S, \text{Sky}) \approx 0.459}$

**Choosing the Splitting Attribute:**

1. Compute $H(S)$ for the current set
2. Compute the weighted entropy after splitting on each candidate attribute

3. The attribute with the **highest** Information Gain is chosen for the node
4. An attribute giving Gain $= 0$ tells us nothing new about the class, so it is never chosen over a better one

### The ID3 Algorithm

**ID3 Algorithm**

**Input:** A set of training examples $S$ a target class attribute and a set of candidate attributes $A$

**Output:** A decision tree that classifies the training examples

1. Create a root node for the tree.
2. If all examples in $S$ belong to the same class $c$ label the root with $c$ and stop.
3. If the attribute set $A$ is empty label the root with the majority class in $S$ and stop.
4. Otherwise compute Information Gain for every remaining attribute in $A$ and select the attribute $A^*$ with the highest gain as the decision attribute for the root.
5. For each possible value $v$ of $A^*$ add a new branch below the root for $A^* = v$ and let $S_v$ be the subset of examples with $A^* = v$.
6. If $S_v$ is empty attach a leaf labeled with the majority class of $S$ below this branch.
7. Otherwise attach the subtree obtained by recursively applying steps 2 to 6 to $S_v$ using $A - \{A^*\}$ as the remaining attribute set.
8. Return the root node.

**Example 2.10 — Building a Tree for the Walk Dataset**

Build the ID3 decision tree for the dataset below.

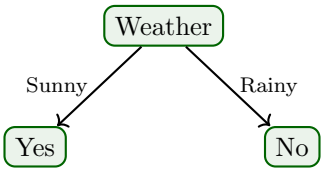
| Weather | Wind | Walk |
|---|---|---|
| Sunny | Weak | Yes |
| Sunny | Weak | Yes |
| Sunny | Strong | Yes |
| Rainy | Weak | No |
| Rainy | Weak | No |
| Rainy | Strong | No |

**Solution:** The set has 3 Yes and 3 No, so $H(S) = 1$ (Example 2.6 pattern).

$$S_{\text{Sunny}} = \{\text{Yes,Yes,Yes}\}: \text{ pure} \Rightarrow H = 0$$
$$S_{\text{Rainy}} = \{\text{No,No,No}\}: \text{ pure} \Rightarrow H = 0$$
$$\text{Gain}(S, \text{Weather}) = 1 - \left[\frac{3}{6}(0) + \frac{3}{6}(0)\right] = 1$$

$$S_{\text{Weak}} = \{\text{Yes,Yes,No,No}\}: \ 2/2 \Rightarrow H = 1$$
$$S_{\text{Strong}} = \{\text{Yes,No}\}: \ 1/1 \Rightarrow H = 1$$
$$\text{Gain}(S, \text{Wind}) = 1 - \left[\frac{4}{6}(1) + \frac{2}{6}(1)\right] = 0$$

Since Gain(Weather) $= 1 >$ Gain(Wind) $= 0$, Weather is chosen as the root. Both resulting subsets are already pure, so ID3 stops immediately (step 2).




Final Tree: Weather $\rightarrow$ {Sunny : Yes, Rainy : No}

**Example 2.11 — Building a Tree for Loan Approval**

Build the ID3 decision tree for the dataset below.

| Income | CreditScore | LoanApproved |
|---|---|---|
| High | Good | Approve |
| High | Bad | Approve |
| High | Bad | Approve |
| Low | Good | Approve |
| Low | Bad | Deny |
| Low | Bad | Deny |

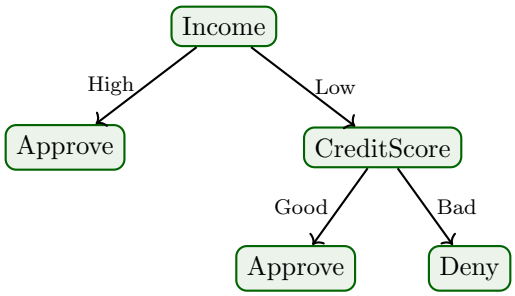
**Solution:** The set has 4 Approve and 2 Deny, so $H(S) \approx 0.918$.

$$S_{\text{High}} = \{\text{Approve,Approve,Approve}\}: \text{ pure} \Rightarrow H = 0$$
$$S_{\text{Low}} = \{\text{Approve,Deny,Deny}\}: 1/3, 2/3 \Rightarrow H \approx 0.918$$
$$\text{Gain}(S, \text{Income}) = 0.918 - \left[\frac{3}{6}(0) + \frac{3}{6}(0.918)\right] = 0.459$$

$$S_{\text{Good}} = \{\text{Approve,Approve}\}: \text{ pure} \Rightarrow H = 0$$
$$S_{\text{Bad}} = \{\text{Approve,Approve,Deny,Deny}\}: 2/2 \Rightarrow H = 1$$
$$\text{Gain}(S, \text{CreditScore}) = 0.918 - \left[\frac{2}{6}(0) + \frac{4}{6}(1)\right] = 0.251$$

Since Gain(Income) $\approx 0.459 >$ Gain(CreditScore) $\approx 0.251$, Income is chosen as the root. Income = High is already pure (Approve). Income = Low is not pure and must be split further using the only remaining attribute, CreditScore:

$$\text{Low, Good} \rightarrow \{\text{Approve}\}: \text{ pure} \Rightarrow \text{leaf} = \text{Approve}$$
$$\text{Low, Bad} \rightarrow \{\text{Deny,Deny}\}: \text{ pure} \Rightarrow \text{leaf} = \text{Deny}$$




Final Tree: Income $\rightarrow$ {High : Approve, Low $\rightarrow$ CreditScore $\rightarrow$ {Good : Approve, Bad : Deny}}

**Example 2.12 — Building a Tree for Exam Result Prediction**

Build the ID3 decision tree for the dataset below.

| Study | Sleep | Result |
|---|---|---|
| Yes | Enough | Pass |
| Yes | Enough | Pass |
| Yes | Little | Pass |
| No | Enough | Fail |
| No | Enough | Fail |
| No | Little | Pass |

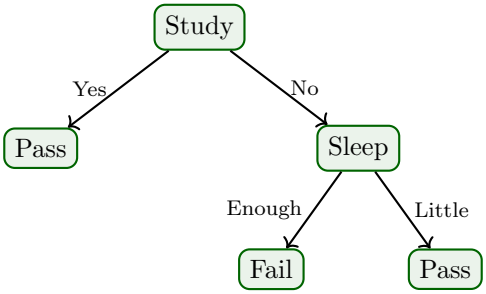
**Solution:** The set has 4 Pass and 2 Fail, so $H(S) \approx 0.918$.

$$S_{\text{Study=Yes}} = \{\text{Pass,Pass,Pass}\}: \text{ pure} \Rightarrow H = 0$$
$$S_{\text{Study=No}} = \{\text{Fail,Fail,Pass}\}: \ 1/3, 2/3 \Rightarrow H \approx 0.918$$
$$\text{Gain}(S, \text{Study}) = 0.918 - \left[\frac{3}{6}(0) + \frac{3}{6}(0.918)\right] = 0.459$$

$$S_{\text{Sleep=Enough}} = \{\text{Pass,Pass,Fail,Fail}\}: \ 2/2 \Rightarrow H = 1$$
$$S_{\text{Sleep=Little}} = \{\text{Pass,Pass}\}: \text{ pure} \Rightarrow H = 0$$
$$\text{Gain}(S, \text{Sleep}) = 0.918 - \left[\frac{4}{6}(1) + \frac{2}{6}(0)\right] = 0.251$$

Since Gain(Study) $\approx 0.459 >$ Gain(Sleep) $\approx 0.251$, Study is chosen as the root. Study = Yes is already pure (Pass). Study = No is not pure and must be split further using the only remaining attribute, Sleep:

$$\text{No, Enough} \to \{\text{Fail,Fail}\}: \text{ pure} \Rightarrow \text{leaf} = \text{Fail}$$
$$\text{No, Little} \to \{\text{Pass}\}: \text{ pure} \Rightarrow \text{leaf} = \text{Pass}$$




Final Tree: Study $\to$ {Yes : Pass, No $\to$ Sleep $\to$ {Enough : Fail, Little : Pass}}

**ID3 Stopping Conditions (Base Cases):**

1. All remaining examples belong to the same class $\Rightarrow$ create a leaf with that class
2. No attributes are left to split on $\Rightarrow$ create a leaf with the majority class
3. A branch has no matching examples $\Rightarrow$ create a leaf with the majority class of the parent set
4. Otherwise pick the attribute with the highest Information Gain and recurse

# Naive Bayes Classifier

## Bayes' Theorem

**Bayes' Theorem**

Bayes' Theorem relates the probability of a hypothesis $H$ given evidence $E$ to the reverse conditional probability:

$$P(H \mid E) = \frac{P(E \mid H)\, P(H)}{P(E)}$$

where $P(H)$ is the **prior**, $P(E \mid H)$ is the **likelihood**, $P(E)$ is the **evidence probability**, and $P(H \mid E)$ is the **posterior**.

This is simply the definition of conditional probability rearranged and applied in the reverse direction.

**Example 2.13 — Medical Test Diagnosis**

Given:

- $P(\text{Disease}) = 0.01$
- $P(\text{Positive} \mid \text{Disease}) = 0.9$
- $P(\text{Positive} \mid \text{NoDisease}) = 0.05$

Find $P(\text{Disease} \mid \text{Positive})$.

**Solution:**

$$\begin{aligned} P(\text{Positive}) &= P(\text{Positive}|\text{Disease})P(\text{Disease}) + P(\text{Positive}|\text{NoDisease})P(\text{NoDisease}) \\ &= (0.9)(0.01) + (0.05)(0.99) \\ &= 0.009 + 0.0495 = 0.0585 \end{aligned}$$

$$P(\text{Disease}|\text{Positive}) = \frac{0.009}{0.0585}$$

$$\boxed{P(\text{Disease} \mid \text{Positive}) \approx 0.154}$$

**Example 2.14 — Choosing a Bag of Balls**

Bag A has 3 red and 2 blue balls. Bag B has 1 red and 4 blue balls. A bag is chosen at random (equal probability) and a red ball is drawn. Find $P(\text{Bag A} \mid \text{Red})$.

**Solution:**

$$\begin{aligned} P(A) = P(B) &= 0.5 \\ P(\text{Red}|A) = \frac{3}{5} = 0.6, &\qquad P(\text{Red}|B) = \frac{1}{5} = 0.2 \\ P(\text{Red}) &= (0.6)(0.5) + (0.2)(0.5) = 0.3 + 0.1 = 0.4 \\ P(A|\text{Red}) &= \frac{0.3}{0.4} \end{aligned}$$

$$\boxed{P(\text{Bag A} \mid \text{Red}) = 0.75}$$

**Example 2.15 — Predicting Rain from Cloud Cover**

Given:

- $P(\text{Rain}) = 0.3$
- $P(\text{Cloudy} \mid \text{Rain}) = 0.8$
- $P(\text{Cloudy} \mid \text{NoRain}) = 0.4$

Find $P(\text{Rain} \mid \text{Cloudy})$.

**Solution:**

$$P(\text{Cloudy}) = (0.8)(0.3) + (0.4)(0.7) = 0.24 + 0.28 = 0.52$$

$$P(\text{Rain}|\text{Cloudy}) = \frac{0.24}{0.52}$$

$$\boxed{P(\text{Rain} \mid \text{Cloudy}) \approx 0.462}$$

## The Naive Bayes Assumption

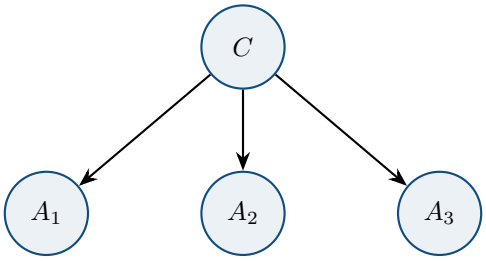
**Naive Bayes Assumption (Conditional Independence)**

Given the class label $C$, the naive Bayes classifier assumes that all attributes $A_1, A_2, \ldots, A_n$ are conditionally independent of one another:

$$P(A_1, A_2, \ldots, A_n \mid C) = \prod_{i=1}^{n} P(A_i \mid C)$$

This means that once the class is known, each attribute's probability can be computed on its own, without needing to know the values of the other attributes.

The diagram below shows this assumption as a simple graphical model: the class $C$ independently influences each attribute, and there are no direct connections between attributes themselves.




This is the same "product of independent probabilities" idea used when multiplying probabilities of independent events.

## Naive Bayes Classification

**Naive Bayes Classification Rule**

1. Compute a score $P(C_k) \prod_i P(A_i \mid C_k)$ for every class $C_k$ *(posterior score)*
2. Predict the class with the **maximum** score *(MAP decision rule)*
3. $P(E)$ is the same for every class comparison, so it can be dropped *(normalization*

*not needed)*

$$\hat{C} = \arg\max_{C_k} P(C_k) \prod_{i=1}^{n} P(A_i \mid C_k)$$

**Naive Bayes Classifier Algorithm**

**Input:** A training set with attributes $A_1, \ldots, A_n$, a class attribute $C$, and one test instance

**Output:** The predicted class label for the test instance

1. For each class $C_k$, compute the prior $P(C_k) = \dfrac{\text{count}(C_k)}{\text{total examples}}$.
2. For each attribute $A_i$ and each class $C_k$, compute the likelihood $P(A_i \mid C_k)$ from the training data.
3. For the test instance, compute the score $P(C_k) \prod_i P(A_i \mid C_k)$ for every class $C_k$ using the instance's attribute values.
4. Predict the class $C_k$ with the highest score.

**Example 2.16 — Predicting PlayCricket**

| Weather | Wind | PlayCricket |
|---|---|---|
| Sunny | Weak | Yes |
| Sunny | Strong | No |
| Rainy | Weak | Yes |
| Rainy | Strong | Yes |
| Sunny | Weak | Yes |
| Rainy | Strong | No |

Classify a new instance: Weather = Sunny Wind = Strong.

**Solution:**

$$P(\text{Yes}) = \frac{4}{6} = \frac{2}{3}, \qquad P(\text{No}) = \frac{2}{6} = \frac{1}{3}$$

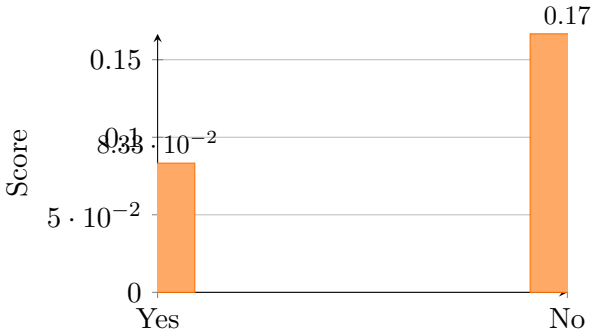
$$P(\text{Sunny}|\text{Yes}) = \frac{2}{4} = 0.5, \qquad P(\text{Strong}|\text{Yes}) = \frac{1}{4} = 0.25$$

$$P(\text{Sunny}|\text{No}) = \frac{1}{2} = 0.5, \qquad P(\text{Strong}|\text{No}) = \frac{2}{2} = 1$$

$$\text{Score}(\text{Yes}) = \frac{2}{3}(0.5)(0.25) \approx 0.0833$$

$$\text{Score}(\text{No}) = \frac{1}{3}(0.5)(1) \approx 0.1667$$

Since Score(No) > Score(Yes): $\boxed{\text{Predicted class: No}}$




**Example 2.17 — Classifying a Fruit**

| Color | Size | Fruit |
|---|---|---|
| Red | Large | Apple |
| Green | Small | Grape |
| Red | Small | Grape |
| Red | Large | Apple |
| Green | Small | Grape |
| Red | Large | Apple |

Classify a new instance: Color = Red Size = Small.

**Solution:**

$$P(\text{Apple}) = \frac{3}{6} = 0.5, \qquad P(\text{Grape}) = \frac{3}{6} = 0.5$$

$$P(\text{Red}|\text{Apple}) = \frac{3}{3} = 1, \qquad P(\text{Small}|\text{Apple}) = \frac{0}{3} = 0$$

$$P(\text{Red}|\text{Grape}) = \frac{1}{3}, \qquad P(\text{Small}|\text{Grape}) = \frac{3}{3} = 1$$

$$\text{Score}(\text{Apple}) = (0.5)(1)(0) = 0$$

$$\text{Score}(\text{Grape}) = (0.5)\left(\frac{1}{3}\right)(1) \approx 0.1667$$

Since Score(Grape) > Score(Apple): $\boxed{\text{Predicted class: Grape}}$

**Example 2.18 — Classifying an Email**

| ContainsMoney | ContainsFree | Class |
|---|---|---|
| Yes | Yes | Spam |
| Yes | No | Spam |
| No | No | NotSpam |
| No | Yes | NotSpam |
| Yes | Yes | Spam |
| No | No | NotSpam |

Classify a new instance: ContainsMoney = Yes ContainsFree = No.

**Solution:**

$$P(\text{Spam}) = \frac{3}{6} = 0.5, \qquad P(\text{NotSpam}) = \frac{3}{6} = 0.5$$
$$P(\text{Money=Yes}|\text{Spam}) = \frac{3}{3} = 1, \qquad P(\text{Free=No}|\text{Spam}) = \frac{1}{3}$$
$$P(\text{Money=Yes}|\text{NotSpam}) = \frac{0}{3} = 0, \qquad P(\text{Free=No}|\text{NotSpam}) = \frac{2}{3}$$

$$\text{Score(Spam)} = (0.5)(1)\left(\frac{1}{3}\right) \approx 0.1667$$
$$\text{Score(NotSpam)} = (0.5)(0)\left(\frac{2}{3}\right) = 0$$

Since Score(Spam) > Score(NotSpam): $\boxed{\text{Predicted class: Spam}}$

**The Zero-Probability Problem:**

1. If an attribute value never occurs with a class in training data, its likelihood is 0
2. A single zero likelihood forces the entire product (and hence the score) to 0, as seen for Apple in Example 2.17
3. This can wrongly rule out a class even when the other evidence supports it
4. Fix: **Laplace smoothing** adds a small count to every likelihood so no probability is ever exactly zero

# Artificial Neural Networks

## The Perceptron Model

**Artificial Neural Network**

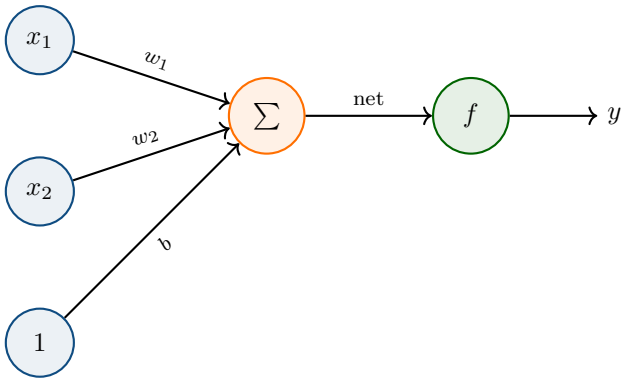
An artificial neural network (ANN) is a computational model inspired by the brain, made of simple processing units called neurons that are connected together. Each connection carries a weight, and the network learns by adjusting these weights so that a given input produces the desired output.

**Perceptron**

A perceptron is the simplest artificial neuron. It takes input values $x_1, x_2, \ldots, x_n$, multiplies each by a weight $w_1, w_2, \ldots, w_n$, adds a bias term $b$, and passes the result through an activation function $f$ to produce the output:

$$\text{net} = \sum_{i=1}^{n} w_i x_i + b, \qquad y = f(\text{net})$$

This is nothing more than a dot product of the weight vector and the input vector, shifted by a constant and passed through a function.




## Activation Functions

**Common Activation Functions**

1. **Step (threshold) function** *(hard decision)*

$$f(\text{net}) = \begin{cases} 1 & \text{if net} \geq 0 \\ 0 & \text{if net} < 0 \end{cases}$$

2. **Sigmoid function** *(smooth 0 to 1 output)*

$$f(\text{net}) = \frac{1}{1 + e^{-\text{net}}}$$

**Perceptron Output Computation**

**Input:** input values $x_1, \ldots, x_n$, weights $w_1, \ldots, w_n$, bias $b$, an activation function $f$
**Output:** the perceptron's output $y$

1. Compute the weighted sum net $= \sum_i w_i x_i + b$.

2. Apply the activation function: $y = f(\text{net})$.
3. Return $y$.

**Example 2.19 — Perceptron with Step Activation**

A perceptron has weights $w_1 = 0.6$, $w_2 = 0.6$, bias $b = -0.5$, and step activation. Compute its output for $x_1 = 1, x_2 = 0$.

**Solution:**

$$\text{net} = (0.6)(1) + (0.6)(0) + (-0.5) = 0.1$$
$$y = f(0.1) = 1 \quad (\text{since } 0.1 \geq 0)$$

$\boxed{y = 1}$

**Example 2.20 — Same Perceptron a Different Input**

Using the same perceptron as Example 2.19, compute the output for $x_1 = 0, x_2 = 0$.

**Solution:**

$$\text{net} = (0.6)(0) + (0.6)(0) + (-0.5) = -0.5$$
$$y = f(-0.5) = 0 \quad (\text{since } -0.5 < 0)$$

$\boxed{y = 0}$

**Example 2.21 — Same Perceptron a Third Input**

Using the same perceptron as Example 2.19, compute the output for $x_1 = 1, x_2 = 1$.

**Solution:**

$$\text{net} = (0.6)(1) + (0.6)(1) + (-0.5) = 0.7$$
$$y = f(0.7) = 1 \quad (\text{since } 0.7 \geq 0)$$

$\boxed{y = 1}$

**Observation from Examples 2.19 to 2.21:**

1. The output is 1 whenever at least one input is 1, and 0 only when both inputs are 0
2. This perceptron has learned the logical **OR** function
3. A single perceptron can only represent **linearly separable** functions (like AND, OR); it cannot represent XOR, which is why hidden layers are needed (Section 2.3.3)

**Example 2.22 — Perceptron with Sigmoid Activation**

A perceptron has weights $w_1 = 0.4$, $w_2 = 0.3$, bias $b = -0.2$, and sigmoid activation. Compute its output for $x_1 = 2, x_2 = 1$. (Use $e^{-0.9} \approx 0.4066$.)

**Solution:**

$$\text{net} = (0.4)(2) + (0.3)(1) + (-0.2) = 0.8 + 0.3 - 0.2 = 0.9$$
$$y = \frac{1}{1 + e^{-0.9}} = \frac{1}{1 + 0.4066} = \frac{1}{1.4066}$$

$\boxed{y \approx 0.711}$

**Example 2.23 — Sigmoid Perceptron a Different Input**

A perceptron has weights $w_1 = 0.5$, $w_2 = -0.5$, bias $b = 0.3$, and sigmoid activation. Compute its output for $x_1 = 0, x_2 = 2$. (Use $e^{0.7} \approx 2.0138$.)

**Solution:**

$$\text{net} = (0.5)(0) + (-0.5)(2) + 0.3 = 0 - 1 + 0.3 = -0.7$$
$$y = \frac{1}{1 + e^{0.7}} = \frac{1}{1 + 2.0138} = \frac{1}{3.0138}$$

$\boxed{y \approx 0.332}$

**Example 2.24 — Sigmoid Perceptron a Third Input**

A perceptron has weights $w_1 = -0.2$, $w_2 = 0.6$, bias $b = 0$, and sigmoid activation. Compute its output for $x_1 = 3, x_2 = 0$. (Use $e^{0.6} \approx 1.8221$.)

**Solution:**

$$\text{net} = (-0.2)(3) + (0.6)(0) + 0 = -0.6$$
$$y = \frac{1}{1 + e^{0.6}} = \frac{1}{1 + 1.8221} = \frac{1}{2.8221}$$

$\boxed{y \approx 0.354}$

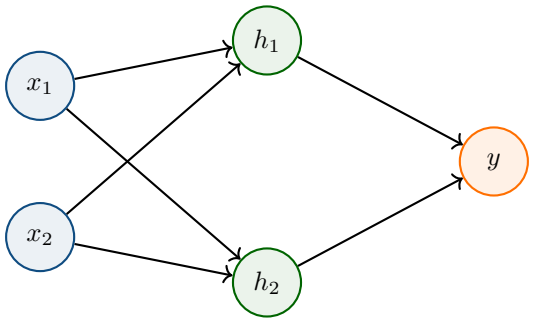
**Step vs Sigmoid:**

1. Step activation gives a hard 0 or 1 output only
2. Sigmoid activation gives a smooth value strictly between 0 and 1, useful as a "confidence" score
3. As net $\to +\infty$, sigmoid $\to 1$; as net $\to -\infty$, sigmoid $\to 0$

## Multilayer Neural Networks and Forward Propagation

**Multilayer Perceptron (MLP)**

A multilayer perceptron is a network with one or more **hidden layers** placed between the input layer and the output layer. Every neuron in one layer connects to every neuron in the next layer, each connection has its own weight, and information flows forward from input to output through the hidden neurons.




Input Layer → Hidden Layer → Output Layer

**Forward Propagation Algorithm**

**Input:** input vector $x_1, \ldots, x_n$, weights and biases for the hidden layer and the output layer, an activation function $f$

**Output:** the network's final output $y$

1. For each hidden neuron $h_j$, compute $\text{net}_{h_j} = \sum_i w_{ij} x_i + b_j$ and then $h_j = f(\text{net}_{h_j})$.
2. Using the hidden outputs $h_j$ as inputs, compute $\text{net}_y = \sum_j v_j h_j + b_{\text{out}}$.
3. Compute the final output $y = f(\text{net}_y)$.
4. Return $y$.

**Example 2.25 — Forward Pass Through a Two Layer Network**

A network has two hidden neurons and one output neuron, all with step activation:

$$h_1 = f(x_1 + x_2 - 0.5), \qquad h_2 = f(x_1 + x_2 - 1.5), \qquad y = f(h_1 - h_2 - 0.5)$$

Compute the output for $x_1 = 1, x_2 = 0$.

**Solution:**

$$\begin{aligned} \text{net}_{h_1} &= 1 + 0 - 0.5 = 0.5 \;\Rightarrow\; h_1 = f(0.5) = 1 \\ \text{net}_{h_2} &= 1 + 0 - 1.5 = -0.5 \;\Rightarrow\; h_2 = f(-0.5) = 0 \\ \text{net}_y &= h_1 - h_2 - 0.5 = 1 - 0 - 0.5 = 0.5 \;\Rightarrow\; y = f(0.5) = 1 \end{aligned}$$

$\boxed{y = 1}$

**Example 2.26 — Forward Pass Through the Same Network**

Using the same network as Example 2.25, compute the output for $x_1 = 1, x_2 = 1$.

**Solution:**

$$\begin{aligned} \text{net}_{h_1} &= 1 + 1 - 0.5 = 1.5 \;\Rightarrow\; h_1 = f(1.5) = 1 \\ \text{net}_{h_2} &= 1 + 1 - 1.5 = 0.5 \;\Rightarrow\; h_2 = f(0.5) = 1 \\ \text{net}_y &= h_1 - h_2 - 0.5 = 1 - 1 - 0.5 = -0.5 \;\Rightarrow\; y = f(-0.5) = 0 \end{aligned}$$

$\boxed{y = 0}$

**Example 2.27 — Forward Pass Through the Same Network**

Using the same network as Example 2.25, compute the output for $x_1 = 0, x_2 = 0$.

**Solution:**

$$\begin{aligned} \text{net}_{h_1} &= 0 + 0 - 0.5 = -0.5 \;\Rightarrow\; h_1 = f(-0.5) = 0 \\ \text{net}_{h_2} &= 0 + 0 - 1.5 = -1.5 \;\Rightarrow\; h_2 = f(-1.5) = 0 \\ \text{net}_y &= h_1 - h_2 - 0.5 = 0 - 0 - 0.5 = -0.5 \;\Rightarrow\; y = f(-0.5) = 0 \end{aligned}$$

$\boxed{y = 0}$

**Why This Network Matters:**

1. Checking all four input combinations gives outputs $0, 1, 1, 0$ for $(0,0), (1,0), (0,1), (1,1)$

2. This is exactly the **XOR** function, which a single perceptron can never compute (see the keypoint after Example 2.21)
3. $h_1$ behaves like an OR gate and $h_2$ behaves like an AND gate; the output combines them as "$h_1$ true AND $h_2$ false"
4. This is the classic illustration of why **hidden layers** give neural networks more power than a single perceptron

# Support Vector Machines

## Linear Separability and the Optimal Hyperplane

**Hyperplane**

A hyperplane is the decision boundary that separates two classes in feature space. In two dimensions it is simply a line, defined by:

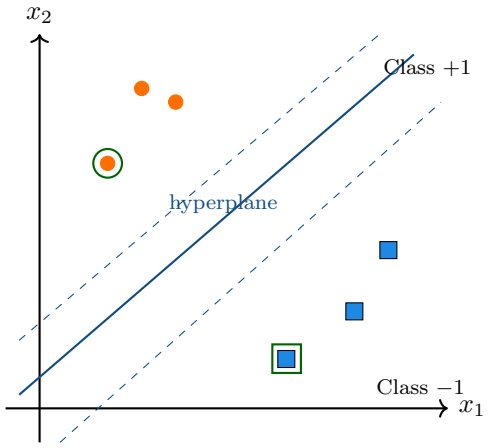
$$w \cdot x + b = 0$$

where $w$ is the weight vector (normal to the hyperplane) and $b$ is the bias term.

**Margin and Support Vectors**

The **margin** is the distance between the hyperplane and the closest training points from either class. The **support vectors** are exactly those closest points — they are the only points that determine the position of the hyperplane. The Support Vector Machine (SVM) chooses the hyperplane that makes this margin as large as possible.

This is the same idea as finding a line that keeps the maximum possible distance from the nearest point on either side, using distance formulas already familiar from coordinate geometry.




The circled points touching the dashed margin lines are the support vectors — every other point could be removed without changing the hyperplane at all.

## The SVM Decision Function

**Properties of the SVM Decision Function**

1. A point is classified using $f(x) = \text{sign}(w \cdot x + b)$ *(decision rule)*
2. $w \cdot x + b > 0$ places the point on the $+1$ side *(positive class)*
3. $w \cdot x + b < 0$ places the point on the $-1$ side *(negative class)*
4. $w \cdot x + b = 0$ means the point lies exactly on the hyperplane *(boundary case)*

**SVM Classification Algorithm**

**Input:** weight vector $w$, bias $b$, and a test point $x$
**Output:** predicted class label $+1$ or $-1$ for $x$

1. Compute the dot product $w \cdot x$.

2. Add the bias: score $= w \cdot x + b$.
3. If score $> 0$ predict class $+1$; if score $< 0$ predict class $-1$; if score $= 0$ the point lies on the hyperplane.
4. Return the predicted class.

**Example 2.28 — Classifying a Point (Positive Side)**

A trained SVM has $w = (2, -1)$ and $b = -1$. Classify the point $x = (3, 1)$.

**Solution:**

$$\begin{aligned} \text{score} &= w \cdot x + b = (2)(3) + (-1)(1) + (-1) \\ &= 6 - 1 - 1 = 4 \end{aligned}$$

Since $4 > 0$: $\boxed{\text{Predicted class: } +1}$

**Example 2.29 — Classifying a Point (Negative Side)**

Using the same SVM as Example 2.28 ($w = (2, -1)$, $b = -1$), classify the point $x = (0, 3)$.

**Solution:**

$$\begin{aligned} \text{score} &= (2)(0) + (-1)(3) + (-1) \\ &= 0 - 3 - 1 = -4 \end{aligned}$$

Since $-4 < 0$: $\boxed{\text{Predicted class: } -1}$

**Example 2.30 — Classifying a Point on the Boundary**

Using the same SVM as Example 2.28 ($w = (2, -1)$, $b = -1$), classify the point $x = (1, 1)$.

**Solution:**

$$\begin{aligned} \text{score} &= (2)(1) + (-1)(1) + (-1) \\ &= 2 - 1 - 1 = 0 \end{aligned}$$

Since score $= 0$, the point lies exactly on the hyperplane, so it is a **support vector**.
$\boxed{x = (1, 1) \text{ lies on the decision boundary}}$

**Reading the Score:**

1. The sign of $w \cdot x + b$ gives the predicted class
2. The magnitude of $w \cdot x + b$ (before normalizing by $\|w\|$) hints at how far the point is from the boundary — larger magnitude means the model is more confident
3. Example 2.30 shows a point exactly on the hyperplane, which is precisely what defines a support vector

## Computing the Margin

**Margin Width Formula**

Once the SVM has found the optimal $w$, the width of the margin (distance between the two dashed boundary lines) is:

$$\text{Margin} = \frac{2}{\|w\|}, \qquad \|w\| = \sqrt{w_1^2 + w_2^2 + \cdots + w_n^2}$$

**Distance from a Point to the Hyperplane**

The perpendicular distance from any point $x$ to the hyperplane $w \cdot x + b = 0$ is:

$$d = \frac{|w \cdot x + b|}{\|w\|}$$

This is simply the point-to-line distance formula the students already know from coordinate geometry, extended to vector notation.

**Example 2.31 — Margin Width for a Given Weight Vector**

An SVM has weight vector $w = (2, -1)$. Compute the margin width.

**Solution:**

$$\|w\| = \sqrt{2^2 + (-1)^2} = \sqrt{4 + 1} = \sqrt{5}$$
$$\text{Margin} = \frac{2}{\sqrt{5}}$$

$\boxed{\text{Margin} \approx 0.894}$

**Example 2.32 — Margin Width for a Simpler Weight Vector**

An SVM has weight vector $w = (3, 4)$. Compute the margin width.

**Solution:**

$$\|w\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$
$$\text{Margin} = \frac{2}{5}$$

$\boxed{\text{Margin} = 0.4}$

**Example 2.33 — Distance from a Point to the Hyperplane**

An SVM has $w = (1, 1)$ and $b = -2$. Find the distance from the point $x = (4, 4)$ to the hyperplane.

**Solution:**

$$w \cdot x + b = (1)(4) + (1)(4) + (-2) = 6$$
$$\|w\| = \sqrt{1^2 + 1^2} = \sqrt{2}$$
$$d = \frac{|6|}{\sqrt{2}}$$

$\boxed{d \approx 4.243}$

**SVM Summary:**

1. A **smaller** $\|w\|$ gives a **larger** margin — this is exactly what SVM training optimizes for
2. Only the support vectors (score $= 0$ direction, i.e. the closest points) affect $w$ and $b$; all other points can change freely without moving the hyperplane
3. For data that is **not** linearly separable in its original features, the **kernel trick** maps points into a higher-dimensional space where a separating hyperplane does exist — the calculations themselves stay the same dot-product-and-sign process shown above, just in the transformed space

# Overfitting Noisy Data and Pruning

## Overfitting in Decision Trees

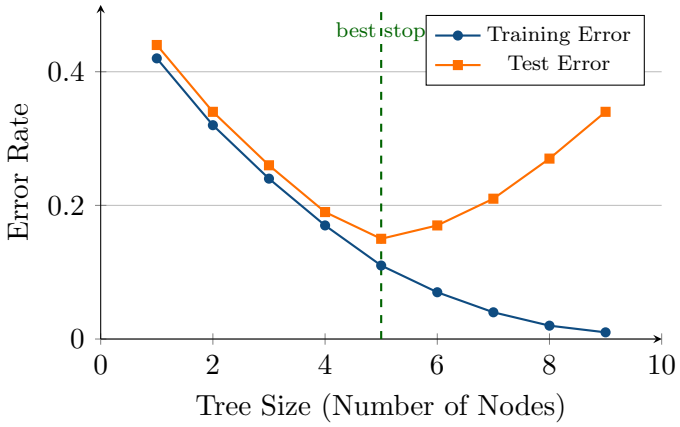
**Overfitting**

A model overfits when it fits the training data too closely including its noise and random fluctuations so that it performs very well on training data but poorly on new unseen data. In decision trees, this happens when the tree grows too many branches trying to perfectly classify every single training example.

**Noise**

Noise refers to errors or random variation in the training data, such as mislabeled examples or incorrect attribute values, that do not reflect the true underlying pattern. A tree that tries to perfectly fit noisy examples ends up learning the noise itself rather than the real pattern.

The graph below shows the classic pattern: as tree size (number of nodes) increases, training error keeps falling, but test error falls only up to a point and then rises again — that turning point is where overfitting begins.




**Example 2.34 — Identifying the Point of Overfitting**

A decision tree is trained to different depths, giving the following accuracies:

| Depth | Training Accuracy | Testing Accuracy |
|---|---|---|
| 2 | 70% | 68% |
| 4 | 85% | 80% |
| 6 | 95% | 78% |
| 8 | 99% | 70% |

At which depth does the tree start to overfit?

**Solution:** Training accuracy keeps rising at every depth (70%,85%,95%,99%), but testing accuracy rises only up to Depth 4 (80%) and then falls at Depth 6 (78%) and Depth 8 (70%). Overfitting begins exactly where testing accuracy turns downward while training accuracy keeps climbing. Overfitting begins after Depth 4 (i.e. from Depth 6 onward)

**Example 2.35 — Choosing the Best Depth to Stop**

Using the table from Example 2.34, which depth should be chosen as the final tree depth?

**Solution:** The best depth is the one with the **highest testing accuracy**, since testing accuracy estimates performance on new data. From the table, the maximum testing accuracy is 80% at Depth 4. $\boxed{\text{Best depth} = 4}$

**Example 2.36 — Computing the Generalization Gap**

Using the table from Example 2.34, compute the generalization gap (Training Accuracy − Testing Accuracy) at each depth and identify which depth is most overfit.

**Solution:**

$$\begin{aligned}
\text{Depth 2:} &\quad 70\% - 68\% = 2\% \\
\text{Depth 4:} &\quad 85\% - 80\% = 5\% \\
\text{Depth 6:} &\quad 95\% - 78\% = 17\% \\
\text{Depth 8:} &\quad 99\% - 70\% = 29\%
\end{aligned}$$

The gap grows steadily and is largest at Depth 8, showing the training accuracy has become disconnected from real performance. $\boxed{\text{Depth 8 is the most overfit (gap} = 29\%)}$

**Signs of Overfitting:**

1. Training accuracy is very high (often near 100%) while testing accuracy is much lower
2. The gap between training and testing accuracy keeps growing as the model grows more complex
3. The tree has very deep branches or leaves covering only one or two training examples

## Pruning

**Pruning**

Pruning is the process of removing branches or subtrees from a decision tree that add complexity without improving (or that even hurt) accuracy on unseen data. **Pre-pruning** stops tree growth early using a stopping condition. **Post-pruning** grows the full tree first and then removes subtrees afterward, checking accuracy on a separate validation set.

This is the same "simplify the model, don't just fit every last training point" idea behind avoiding an unnecessarily high-degree curve fit in regression.

**Reduced Error Pruning**

**Input:** a fully grown decision tree $T$, a validation dataset $V$ separate from the training data

**Output:** a pruned decision tree $T'$

1. Compute the accuracy of $T$ on $V$.
2. For each internal node $n$ (starting near the leaves and working upward), temporarily replace the subtree rooted at $n$ with a leaf labeled by the majority class of the training examples reaching $n$.
3. Evaluate the accuracy of this modified tree on $V$.
4. If the accuracy on $V$ does not decrease, keep the replacement permanently (prune the subtree); otherwise restore the original subtree.

5. Repeat steps 2 to 4 for every internal node until no further pruning improves or maintains validation accuracy.
6. Return the final pruned tree $T'$.

**Example 2.37 — Deciding Whether to Prune (Accuracy Improves)**

Before pruning a subtree at node $X$, validation accuracy is 82%. After replacing the subtree at $X$ with a leaf, validation accuracy becomes 85%. Should node $X$'s subtree be pruned?

**Solution:** Compare accuracies: $85\% > 82\%$, so accuracy does not decrease after pruning; in fact it improves. Yes, prune the subtree at $X$

**Example 2.38 — Deciding Whether to Prune (Accuracy Drops)**

Before pruning a subtree at node $Y$, validation accuracy is 90%. After replacing the subtree at $Y$ with a leaf, validation accuracy becomes 88%. Should node $Y$'s subtree be pruned?

**Solution:** Compare accuracies: $88\% < 90\%$, so accuracy decreases after pruning. No, keep the subtree at Y

**Example 2.39 — Deciding Whether to Prune (Accuracy Unchanged)**

Before pruning a subtree at node $Z$, validation accuracy is 78%. After replacing the subtree at $Z$ with a leaf, validation accuracy remains 78%. Should node $Z$'s subtree be pruned?

**Solution:** Compare accuracies: $78\% = 78\%$, accuracy does not decrease. Reduced Error Pruning prefers the simpler tree whenever accuracy does not get worse. Yes, prune the subtree at $Z$

**Pre-Pruning vs Post-Pruning:**

1. **Pre-pruning** is faster since the tree never grows too large, but risks stopping too early (underfitting)
2. **Post-pruning** is more reliable since it evaluates the effect of removing each subtree directly on validation data
3. Pruning always trades a small amount of training accuracy for better generalization to new data

# Measuring Classifier Accuracy

## The Confusion Matrix

**Confusion Matrix**

A confusion matrix is a table that summarizes a classifier's predictions against the true labels for a binary classification problem. It has four counts: **True Positives (TP)**, **False Positives (FP)**, **False Negatives (FN)**, and **True Negatives (TN)**.

| | | **Actual** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted** | Positive | TP | FP |
| | Negative | FN | TN |

## Accuracy, Precision, Recall, and F1-Score

**Classifier Evaluation Metrics**

1. Accuracy $= \dfrac{TP + TN}{TP + TN + FP + FN}$ *(overall correctness)*
2. Precision $= \dfrac{TP}{TP + FP}$ *(of predicted positives, how many were right)*
3. Recall $= \dfrac{TP}{TP + FN}$ *(of actual positives, how many were found)*
4. F1-score $= \dfrac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$ *(harmonic mean of Precision and Recall)*

**Computing Classifier Metrics**

**Input:** confusion matrix counts $TP$, $FP$, $FN$, $TN$
**Output:** Accuracy, Precision, Recall, F1-score

1. Compute Accuracy $= \dfrac{TP + TN}{TP + TN + FP + FN}$.
2. Compute Precision $= \dfrac{TP}{TP + FP}$.
3. Compute Recall $= \dfrac{TP}{TP + FN}$.
4. Compute F1-score $= \dfrac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.
5. Return all four values.

**Example 2.40 — Metrics from a Balanced Confusion Matrix**

A classifier gives the confusion matrix below.

| | | Actual | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted | Positive | 40 | 10 |
| | Negative | 5 | 45 |

Compute Accuracy, Precision, Recall, and F1-score.

**Solution:**

$$\text{Accuracy} = \frac{40+45}{40+45+10+5} = \frac{85}{100} = 0.85$$
$$\text{Precision} = \frac{40}{40+10} = \frac{40}{50} = 0.8$$
$$\text{Recall} = \frac{40}{40+5} = \frac{40}{45} \approx 0.889$$
$$\text{F1} = \frac{2(0.8)(0.889)}{0.8+0.889} = \frac{1.422}{1.689}$$

$\boxed{\text{Accuracy} = 0.85,\ \text{Precision} = 0.8,\ \text{Recall} \approx 0.889,\ F1 \approx 0.842}$

**Example 2.41 — Metrics from a Small Balanced Matrix**

A classifier gives the confusion matrix below.

| | | Actual | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted | Positive | 8 | 2 |
| | Negative | 2 | 8 |

Compute Accuracy, Precision, Recall, and F1-score.

**Solution:**

$$\text{Accuracy} = \frac{8+8}{8+8+2+2} = \frac{16}{20} = 0.8$$
$$\text{Precision} = \frac{8}{8+2} = 0.8$$
$$\text{Recall} = \frac{8}{8+2} = 0.8$$
$$\text{F1} = \frac{2(0.8)(0.8)}{0.8+0.8} = \frac{1.28}{1.6}$$

$\boxed{\text{Accuracy} = \text{Precision} = \text{Recall} = F1 = 0.8}$

**Example 2.42 — Metrics from an Imbalanced Confusion Matrix**

A classifier gives the confusion matrix below.

| | | Actual | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted | Positive | 5 | 15 |
| | Negative | 0 | 80 |

Compute Accuracy, Precision, Recall, and F1-score.

**Solution:**

$$\text{Accuracy} = \frac{5+80}{5+80+15+0} = \frac{85}{100} = 0.85$$
$$\text{Precision} = \frac{5}{5+15} = \frac{5}{20} = 0.25$$
$$\text{Recall} = \frac{5}{5+0} = \frac{5}{5} = 1.0$$
$$\text{F1} = \frac{2(0.25)(1.0)}{0.25+1.0} = \frac{0.5}{1.25}$$

$\boxed{\text{Accuracy} = 0.85,\ \text{Precision} = 0.25,\ \text{Recall} = 1.0,\ F1 = 0.4}$ Although accuracy looks high (85%), the very low precision reveals the classifier is predicting "Positive" far too often — this is exactly why accuracy alone can be misleading. ✓

**Choosing the Right Metric:**

1. **Accuracy** can be misleading when classes are imbalanced, as seen in Example 2.42
2. **Precision** matters most when false positives are costly (e.g. spam filtering)
3. **Recall** matters most when false negatives are costly (e.g. disease detection)
4. **F1-score** is used when a single balance between Precision and Recall is needed

# Linear Regression

## The Simple Linear Regression Model

**Linear Regression**

Linear regression models the relationship between one input attribute $x$ and a continuous output $y$ by fitting a straight line:

$$\hat{y} = b_0 + b_1 x$$

where $b_1$ is the slope (how much $y$ changes per unit change in $x$) and $b_0$ is the intercept (the predicted value when $x = 0$).

**Least Squares Method**

The best-fitting line is the one that minimizes the sum of squared differences between the actual values $y_i$ and the predicted values $\hat{y}_i$. The coefficients that achieve this minimum are:

$$b_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}, \qquad b_0 = \bar{y} - b_1 \bar{x}$$

where $\bar{x}$ and $\bar{y}$ are the means of the $x$ and $y$ values.

This is the same mean-and-deviation idea used when computing variance, just applied to two variables together instead of one.

**Simple Linear Regression (Least Squares)**

**Input:** training points $(x_1, y_1), \ldots, (x_n, y_n)$, and a new value $x_{\text{new}}$ to predict
**Output:** coefficients $b_0$, $b_1$, and the predicted value $\hat{y}_{\text{new}}$

1. Compute the means $\bar{x} = \frac{1}{n}\sum_i x_i$ and $\bar{y} = \frac{1}{n}\sum_i y_i$.
2. Compute $\sum_i (x_i - \bar{x})(y_i - \bar{y})$ and $\sum_i (x_i - \bar{x})^2$.
3. Compute the slope $b_1 = \dfrac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$.
4. Compute the intercept $b_0 = \bar{y} - b_1 \bar{x}$.
5. Form the regression equation $\hat{y} = b_0 + b_1 x$.
6. Substitute $x_{\text{new}}$ into the equation to get $\hat{y}_{\text{new}}$.

**Example 2.43 — Predicting Exam Score from Study Hours**

| Hours Studied ($x$) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Score /10 ($y$) | 2 | 3 | 5 | 4 | 6 |

Find the regression line and predict the score for a student who studies 6 hours.

**Solution:**

$$\bar{x} = \frac{1+2+3+4+5}{5} = 3, \qquad \bar{y} = \frac{2+3+5+4+6}{5} = 4$$

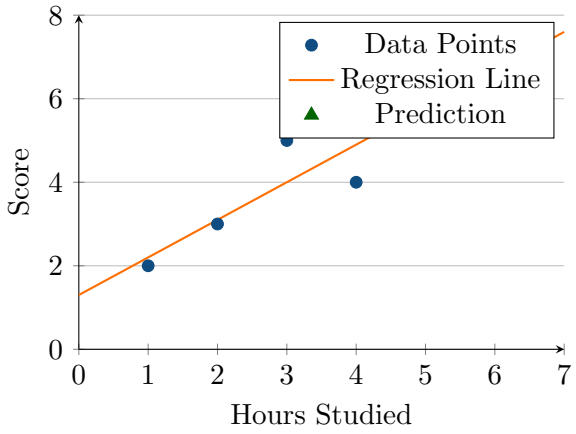
$$\sum (x_i - \bar{x})(y_i - \bar{y}) = (-2)(-2) + (-1)(-1) + (0)(1) + (1)(0) + (2)(2) = 4 + 1 + 0 + 0 + 4 = 9$$

$$\sum (x_i - \bar{x})^2 = 4 + 1 + 0 + 1 + 4 = 10$$

$$b_1 = \frac{9}{10} = 0.9, \qquad b_0 = 4 - (0.9)(3) = 1.3$$

$$\hat{y} = 1.3 + 0.9x$$

Predicting for $x = 6$: $\hat{y} = 1.3 + 0.9(6) = 1.3 + 5.4$ $\boxed{\hat{y}(6) = 6.7}$




**Example 2.44 — Predicting Sales from Advertising Spend**

| Spend ($1000s) ($x$) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Units Sold (100s) ($y$) | 3 | 4 | 2 | 5 | 6 |

Find the regression line and predict units sold for a spend of 7 ($1000s).

**Solution:**

$$\bar{x} = 3, \qquad \bar{y} = \frac{3+4+2+5+6}{5} = 4$$

$$\sum (x_i - \bar{x})(y_i - \bar{y}) = (-2)(-1) + (-1)(0) + (0)(-2) + (1)(1) + (2)(2) = 2 + 0 + 0 + 1 + 4 = 7$$

$$\sum (x_i - \bar{x})^2 = 10$$

$$b_1 = \frac{7}{10} = 0.7, \qquad b_0 = 4 - (0.7)(3) = 1.9$$

$$\hat{y} = 1.9 + 0.7x$$

Predicting for $x = 7$: $\hat{y} = 1.9 + 0.7(7) = 1.9 + 4.9$ $\boxed{\hat{y}(7) = 6.8}$

**Example 2.45 — Predicting Plant Height Growth**

| Weeks Grown ($x$) | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Height Growth cm ($y$) | 1 | 2 | 2 | 3 |

Find the regression line and predict the growth after 5 weeks.

**Solution:**

$$\bar{x} = \frac{1+2+3+4}{4} = 2.5, \qquad \bar{y} = \frac{1+2+2+3}{4} = 2$$

$$\sum (x_i - \bar{x})(y_i - \bar{y}) = (-1.5)(-1) + (-0.5)(0) + (0.5)(0) + (1.5)(1) = 1.5 + 0 + 0 + 1.5 = 3$$

$$\sum (x_i - \bar{x})^2 = 2.25 + 0.25 + 0.25 + 2.25 = 5$$

$$b_1 = \frac{3}{5} = 0.6, \qquad b_0 = 2 - (0.6)(2.5) = 0.5$$

$$\hat{y} = 0.5 + 0.6x$$

Predicting for $x = 5$: $\hat{y} = 0.5 + 0.6(5) = 0.5 + 3$ $\boxed{\hat{y}(5) = 3.5}$

**Reading a Regression Line:**

1. $b_1$ (slope) tells us how much $y$ increases for every one-unit increase in $x$
2. $b_0$ (intercept) is only meaningful as the predicted $y$ when $x = 0$
3. Predictions far outside the range of the original $x$ values (extrapolation) are less reliable

# Logistic Regression

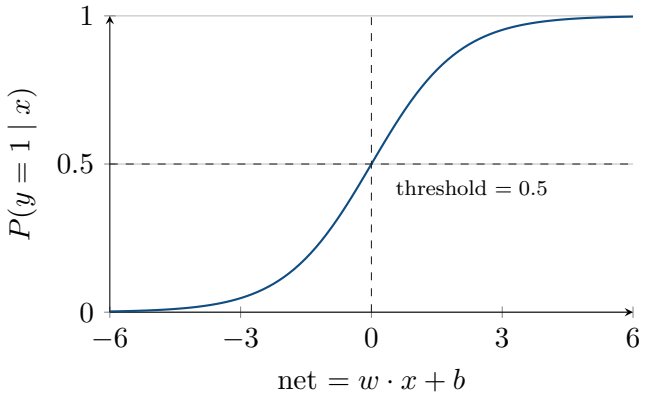
## Modeling Probability with the Sigmoid Function

**Logistic Regression**

Logistic regression is a classification algorithm (despite the name) that models the probability of a binary class label $y \in \{0, 1\}$ using the sigmoid of a linear combination of the attributes:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

This is the same weighted-sum idea used in the perceptron (Section 2.3.1), but the sigmoid output is now interpreted directly as a probability rather than just a squashed activation value.




## The Logistic Regression Decision Rule

**Logistic Regression Decision Rule**

1. If $P(y = 1 \mid x) \geq 0.5$ predict class 1 otherwise predict class 0 *(threshold rule)*
2. $P(y = 1 \mid x) \geq 0.5$ is equivalent to net $= w \cdot x + b \geq 0$ *(shortcut check)*
3. The decision boundary is the hyperplane $w \cdot x + b = 0$ *(same linear boundary idea as SVM)*

**Logistic Regression Classification**

**Input:** weights $w$, bias $b$, a test point $x$, decision threshold (default 0.5)
**Output:** predicted class label (0 or 1) and probability $P$

1. Compute net $= w \cdot x + b$.
2. Compute $P = \dfrac{1}{1 + e^{-\text{net}}}$.
3. If $P \geq 0.5$ predict class 1; otherwise predict class 0.
4. Return the predicted class and $P$.

**Example 2.46 — Logistic Regression Prediction (Net Positive)**

A model has $w = (0.8, -0.5)$, $b = -0.1$. Classify $x = (2, 1)$. (Use $e^{-1} \approx 0.3679$.)

**Solution:**

$$\text{net} = (0.8)(2) + (-0.5)(1) + (-0.1) = 1.6 - 0.5 - 0.1 = 1.0$$
$$P = \frac{1}{1+e^{-1}} = \frac{1}{1+0.3679} = \frac{1}{1.3679}$$
$$P \approx 0.731$$

Since $P \geq 0.5$: $\boxed{\text{Predicted class} = 1}$

**Example 2.47 — Logistic Regression Prediction (Net Negative)**

A model has $w = (0.3, 0.3)$, $b = -1.5$. Classify $x = (1, 1)$. (Use $e^{0.9} \approx 2.4596$.)

**Solution:**

$$\text{net} = (0.3)(1) + (0.3)(1) + (-1.5) = 0.3 + 0.3 - 1.5 = -0.9$$
$$P = \frac{1}{1+e^{0.9}} = \frac{1}{1+2.4596} = \frac{1}{3.4596}$$
$$P \approx 0.289$$

Since $P < 0.5$: $\boxed{\text{Predicted class} = 0}$

**Example 2.48 — Logistic Regression Prediction (Net Zero)**

A model has $w = (1, -1)$, $b = 0$. Classify $x = (2, 2)$.

**Solution:**

$$\text{net} = (1)(2) + (-1)(2) + 0 = 2 - 2 = 0$$
$$P = \frac{1}{1+e^{0}} = \frac{1}{1+1} = 0.5$$

Since $P = 0.5$ meets the threshold $\geq 0.5$: $\boxed{\text{Predicted class} = 1 \text{ (point lies exactly on the decision boundary)}}$

**Linear Regression vs Logistic Regression:**

1. Linear regression predicts a **continuous** value directly as $b_0 + b_1 x$
2. Logistic regression predicts a **probability** between 0 and 1 by passing $w \cdot x + b$ through the sigmoid function
3. Both are built from the same linear combination of attributes; only the final step (identity vs sigmoid) differs
4. Logistic regression is used for classification, not for predicting numeric quantities

## Practice Problems

1. A dataset $S$ of 8 training examples has 5 examples labeled Win and 3 examples labeled Lose. Compute the entropy $H(S)$.

2. A dataset of 8 examples for predicting Class based on Humidity is given below.

| Humidity | Class |
|---|---|
| High | No |
| High | No |
| High | Yes |
| Normal | Yes |
| Normal | Yes |
| Normal | No |
| High | No |
| Normal | Yes |

   Compute Gain$(S, \text{Humidity})$.

3. Using the ID3 algorithm, determine the root attribute and construct the complete decision tree for the dataset below.

| Outlook | Wind | Play |
|---|---|---|
| Overcast | Weak | Yes |
| Overcast | Strong | Yes |
| Overcast | Weak | Yes |
| Sunny | Weak | Yes |
| Sunny | Strong | No |
| Sunny | Strong | No |

4. Using Naive Bayes, classify a new instance with Cough = No Fever = Yes given the training data below.

| Cough | Fever | Flu |
|---|---|---|
| Yes | Yes | Yes |
| Yes | No | Yes |
| No | Yes | No |
| No | No | No |
| Yes | Yes | Yes |
| No | No | No |

5. A bank uses fraud-alert software where $P(\text{Fraud}) = 0.02$, $P(\text{Alert} \mid \text{Fraud}) = 0.95$, and $P(\text{Alert} \mid \text{NoFraud}) = 0.1$. Using Bayes' Theorem, find $P(\text{Fraud} \mid \text{Alert})$.

6. A perceptron has weights $w_1 = 1$, $w_2 = 1$, bias $b = -1.5$, and step activation. Compute its output for all four input combinations $(0,0), (0,1), (1,0), (1,1)$, and state which logical function this perceptron represents.

7. A two-layer network with step activation is defined by $h_1 = f(x_1 + x_2 - 1.5)$, $h_2 = f(x_1 - x_2 - 0.5)$, and $y = f(h_1 + h_2 - 0.5)$. Compute the network's output for $x_1 = 1, x_2 = 1$.

8. An SVM has weight vector $w = (1, 2)$ and bias $b = -3$. Classify the points $x = (2, 1)$ and $x = (0, 0)$.

9. An SVM has weight vector $w = (3, -4)$. Compute the margin width. If the bias is $b = -5$, also compute the perpendicular distance from the point $x = (2, 3)$ to the hyperplane.

10. A decision tree is trained to different depths, giving the accuracies below.

| Depth | Training Accuracy | Testing Accuracy |
|---|---|---|
| 2 | 65% | 63% |
| 4 | 80% | 77% |
| 6 | 93% | 74% |
| 8 | 98% | 65% |

Identify the depth at which the tree begins to overfit and state which depth should be chosen as final.

11. Before pruning a subtree at node $M$, validation accuracy is 84%. After replacing the subtree with a leaf, validation accuracy remains 84%. Using the Reduced Error Pruning rule, should the subtree at $M$ be pruned? Justify your answer.

12. A classifier produces $TP = 30$, $FP = 5$, $FN = 10$, $TN = 55$. Compute Accuracy, Precision, Recall, and F1-score.

13. A classifier produces $TP = 2$, $FP = 1$, $FN = 18$, $TN = 79$. Compute Accuracy, Precision, Recall, and F1-score, and briefly state which metric best reveals a weakness in this classifier.

14. The table below shows advertisement budget and units sold.

| Budget (\$100s) ($x$) | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Units Sold ($y$) | 3 | 4 | 6 | 5 | 7 |

Find the regression line and predict units sold for a budget of 7 (\$100s).

15. A logistic regression model has $w = (0.5, 0.5)$ and $b = -1.2$. Compute net, the probability $P(y = 1 \mid x)$, and the predicted class for $x = (1, 1)$. (Use $e^{0.2} \approx 1.2214$.)

# Chapter 3

# Unsupervised and Instance-Based Learning

## Hierarchical Agglomerative Clustering

### Introduction to Clustering

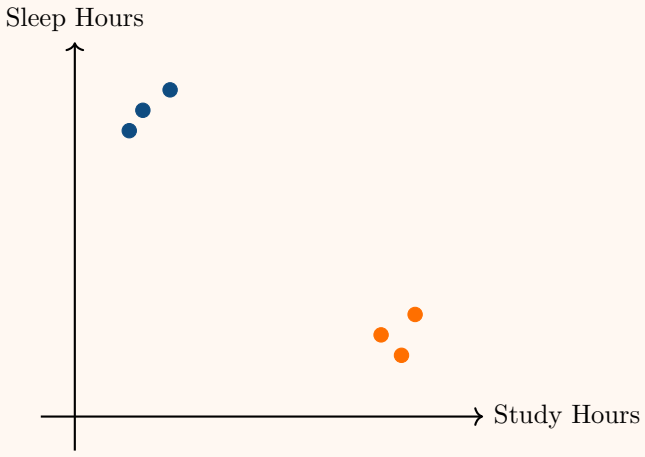
**Clustering**

Clustering is an **unsupervised learning** task in which examples are grouped into clusters based purely on how similar or close they are to one another, without using any class labels. Unlike classification, the algorithm is never told the correct group for any example; it must discover the natural groupings on its own using distance or similarity.

This is the opposite setting from Naive Bayes or Decision Trees, where every training example came with a known class label attached.

**Example 3.1 — Identifying Clusters by Eye**

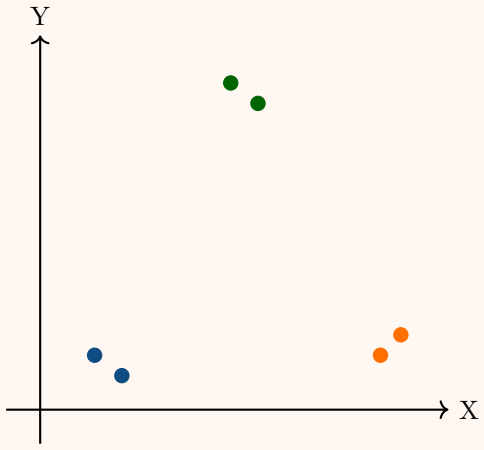
The scatter plot below shows 6 students plotted by (Study Hours, Sleep Hours). By visual inspection, how many natural groups do you see?




**Solution:** The three blue points near the top-left are all close to one another, and the three orange points near the bottom-right are all close to one another, but the two groups are far apart from each other. 2 clusters: {top-left group}, {bottom-right group}

**Example 3.2 — Identifying Three Clusters**

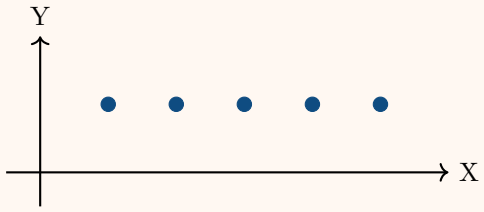
The scatter plot below shows 6 shops plotted by (Location X, Location Y). How many natural groups do you see?




**Solution:** Three separate pairs of points are visible, each pair close together but far from the other pairs: one pair near the bottom-left, one pair near the bottom-right, and one pair near the top-middle. 3 clusters, 2 points each

**Example 3.3 — No Clear Separation**

The scatter plot below shows 5 points spaced almost evenly along a line. How many natural clusters do you see?




**Solution:** Every point is roughly equally spaced from its neighbours, so there is no clear gap separating one group from another. No clear natural clusters — the points form one continuous chain

**Supervised vs Unsupervised Learning:**

1. **Supervised** (Decision Trees, Naive Bayes, SVM, Regression): training data has known class labels
2. **Unsupervised** (Clustering): training data has **no** labels; the algorithm groups examples by similarity alone
3. A good clustering keeps points within the same cluster close together while keeping different clusters far apart, as seen in Examples 3.1 and 3.2

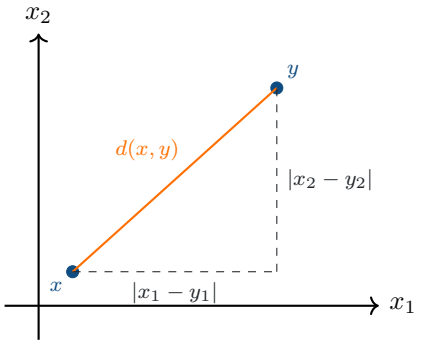
## Distance Measures

**Euclidean Distance**

The Euclidean distance between two points $x = (x_1, x_2)$ and $y = (y_1, y_2)$ measures straight-line closeness:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

This is exactly the Pythagorean theorem: the distance is the hypotenuse of a right triangle formed by the horizontal gap and the vertical gap between the two points.




**Example 3.4 — Distance Between Two Simple Points**

Compute the Euclidean distance between $(1, 2)$ and $(4, 6)$.

**Solution:**

$$d = \sqrt{(1-4)^2 + (2-6)^2} = \sqrt{(-3)^2 + (-4)^2} = \sqrt{9 + 16} = \sqrt{25}$$

$\boxed{d = 5}$

**Example 3.5 — Distance from the Origin**

Compute the Euclidean distance between $(0, 0)$ and $(6, 8)$.

**Solution:**

$$d = \sqrt{(0-6)^2 + (0-8)^2} = \sqrt{36 + 64} = \sqrt{100}$$

$\boxed{d = 10}$

**Example 3.6 — Distance Along a Vertical Line**

Compute the Euclidean distance between $(2, 3)$ and $(2, 7)$.

**Solution:**

$$d = \sqrt{(2-2)^2 + (3-7)^2} = \sqrt{0 + 16} = \sqrt{16}$$

$\boxed{d = 4}$

**Distance Facts:**

1. $d(x, y) = 0$ only when $x$ and $y$ are the same point
2. $d(x, y) = d(y, x)$ always *(symmetry)*
3. Smaller distance means the two points are more similar; clustering always groups the

smallest-distance points first

## Linkage Methods Between Clusters

**Linkage Distance Formulas**

Given two clusters $A$ and $B$, the distance between them can be defined in three ways:

1. **Single Linkage:** $d(A,B) = \min_{a\in A,\, b\in B} d(a,b)$ *(closest pair)*
2. **Complete Linkage:** $d(A,B) = \max_{a\in A,\, b\in B} d(a,b)$ *(farthest pair)*
3. **Average Linkage:** $d(A,B) = \frac{1}{|A||B|} \sum_{a\in A,\, b\in B} d(a,b)$ *(mean of all pairs)*

**Example 3.7 — Single Linkage Between Two Clusters**

Cluster $A = \{(1,1),(3,2)\}$ and Cluster $B = \{(6,1),(7,3)\}$. Compute the single linkage distance $d(A,B)$.

**Solution:**

$$d((1,1),(6,1)) = \sqrt{25+0} = 5$$
$$d((1,1),(7,3)) = \sqrt{36+4} = \sqrt{40} \approx 6.325$$
$$d((3,2),(6,1)) = \sqrt{9+1} = \sqrt{10} \approx 3.162$$
$$d((3,2),(7,3)) = \sqrt{16+1} = \sqrt{17} \approx 4.123$$

Single linkage takes the **minimum** of these four values: $\boxed{d(A,B) \approx 3.162}$

**Example 3.8 — Complete Linkage Between Two Clusters**

Cluster $A = \{(0,0),(1,2)\}$ and Cluster $B = \{(4,0),(5,3)\}$. Compute the complete linkage distance $d(A,B)$.

**Solution:**

$$d((0,0),(4,0)) = 4$$
$$d((0,0),(5,3)) = \sqrt{25+9} = \sqrt{34} \approx 5.831$$
$$d((1,2),(4,0)) = \sqrt{9+4} = \sqrt{13} \approx 3.606$$
$$d((1,2),(5,3)) = \sqrt{16+1} = \sqrt{17} \approx 4.123$$

Complete linkage takes the **maximum** of these four values: $\boxed{d(A,B) \approx 5.831}$

**Example 3.9 — Average Linkage Between Two Clusters**

Cluster $A = \{(2,2),(3,4)\}$ and Cluster $B = \{(6,2),(7,5)\}$. Compute the average linkage distance $d(A,B)$.

**Solution:**

$$d((2,2),(6,2)) = 4$$
$$d((2,2),(7,5)) = \sqrt{25+9} = \sqrt{34} \approx 5.831$$
$$d((3,4),(6,2)) = \sqrt{9+4} = \sqrt{13} \approx 3.606$$
$$d((3,4),(7,5)) = \sqrt{16+1} = \sqrt{17} \approx 4.123$$
$$d(A,B) = \frac{4 + 5.831 + 3.606 + 4.123}{4} = \frac{17.560}{4}$$

$\boxed{d(A,B) \approx 4.390}$

## The Agglomerative Clustering Algorithm

**Hierarchical Agglomerative Clustering**

Agglomerative clustering builds a hierarchy of clusters bottom-up: it starts with every point as its own cluster, and repeatedly merges the two closest clusters (using a chosen linkage rule) until only one cluster remains. The resulting merge history is displayed as a tree diagram called a **dendrogram**, where the height of each merge shows the distance at which it happened.

**Agglomerative Clustering Algorithm**

**Input:** a set of $n$ points, a linkage method (single, complete, or average)
**Output:** a dendrogram showing the full merge history

1. Start with $n$ clusters, one for each point.
2. Compute the pairwise distance between every pair of clusters.
3. Find the two clusters with the smallest distance and merge them into one cluster.
4. Record the distance at which this merge happened (this becomes a height on the dendrogram).
5. Recompute the distance from the newly merged cluster to every remaining cluster using the chosen linkage rule.
6. Repeat steps 3 to 5 until only one cluster remains.
7. Return the full sequence of merges as a dendrogram.

**Example 3.10 — Full Trace Using Single Linkage**

Cluster the points $P_1 = (1,1)$, $P_2 = (2,1)$, $P_3 = (5,1)$, $P_4 = (6.5,1)$ using single linkage. Show every merge and draw the dendrogram.

**Solution:** Since all points lie on the line $y = 1$, distance is just the difference in $x$-coordinates.

| | $P_1$ | $P_2$ | $P_3$ | $P_4$ |
|---|---|---|---|---|
| $P_1$ | – | 1 | 4 | 5.5 |
| $P_2$ | 1 | – | 3 | 4.5 |
| $P_3$ | 4 | 3 | – | 1.5 |
| $P_4$ | 5.5 | 4.5 | 1.5 | – |

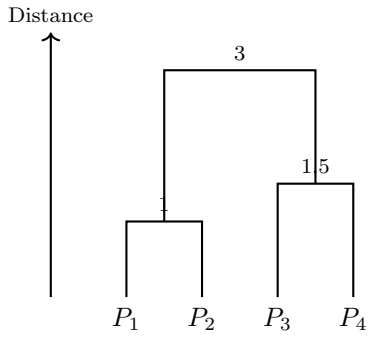
**Step 1:** Smallest distance is $d(P_1, P_2) = 1$. Merge into $\{P_1, P_2\}$.

$$d(\{P_1, P_2\}, P_3) = \min(4, 3) = 3$$
$$d(\{P_1, P_2\}, P_4) = \min(5.5, 4.5) = 4.5$$
$$d(P_3, P_4) = 1.5$$

**Step 2:** Smallest distance is now $d(P_3, P_4) = 1.5$. Merge into $\{P_3, P_4\}$.

$$d(\{P_1, P_2\}, \{P_3, P_4\}) = \min(4, 5.5, 3, 4.5) = 3$$

**Step 3:** Only two clusters remain; merge them at height 3.




Merge order: $(P_1, P_2)$ at 1, $(P_3, P_4)$ at 1.5, root at 3

**Example 3.11 — Full Trace Using Complete Linkage**

Cluster the points $Q_1 = (1, 1)$, $Q_2 = (2, 3)$, $Q_3 = (6, 2)$, $Q_4 = (7, 5)$ using complete linkage. Show every merge and draw the dendrogram.

**Solution:**

$$d(Q_1, Q_2) = \sqrt{1+4} \approx 2.236, \qquad d(Q_1, Q_3) = \sqrt{25+1} \approx 5.099$$
$$d(Q_1, Q_4) = \sqrt{36+16} \approx 7.211, \qquad d(Q_2, Q_3) = \sqrt{16+1} \approx 4.123$$
$$d(Q_2, Q_4) = \sqrt{25+4} \approx 5.385, \qquad d(Q_3, Q_4) = \sqrt{1+9} \approx 3.162$$

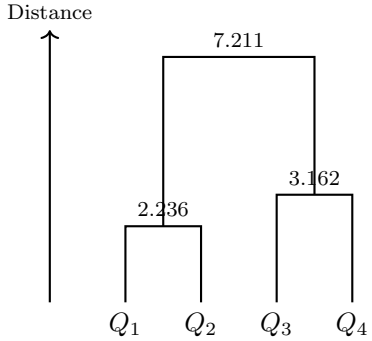
**Step 1:** Smallest distance is $d(Q_1, Q_2) \approx 2.236$. Merge into $\{Q_1, Q_2\}$.

$$d(\{Q_1, Q_2\}, Q_3) = \max(5.099, 4.123) = 5.099$$
$$d(\{Q_1, Q_2\}, Q_4) = \max(7.211, 5.385) = 7.211$$
$$d(Q_3, Q_4) = 3.162$$

**Step 2:** Smallest distance is now $d(Q_3, Q_4) \approx 3.162$. Merge into $\{Q_3, Q_4\}$.

$$d(\{Q_1, Q_2\}, \{Q_3, Q_4\}) = \max(5.099, 7.211, 4.123, 5.385) = 7.211$$

**Step 3:** Only two clusters remain; merge them at height $\approx 7.211$.



Merge order: $(Q_1, Q_2)$ at 2.236, $(Q_3, Q_4)$ at 3.162, root at 7.211

**Example 3.12 — Full Trace Using Average Linkage**

Cluster the points $R_1 = (0,0)$, $R_2 = (1,1)$, $R_3 = (4,0)$, $R_4 = (5,2)$ using average linkage. Show every merge and draw the dendrogram.

**Solution:**

$$d(R_1, R_2) = \sqrt{1+1} \approx 1.414, \qquad d(R_1, R_3) = 4$$
$$d(R_1, R_4) = \sqrt{25+4} \approx 5.385, \qquad d(R_2, R_3) = \sqrt{9+1} \approx 3.162$$
$$d(R_2, R_4) = \sqrt{16+1} \approx 4.123, \qquad d(R_3, R_4) = \sqrt{1+4} \approx 2.236$$

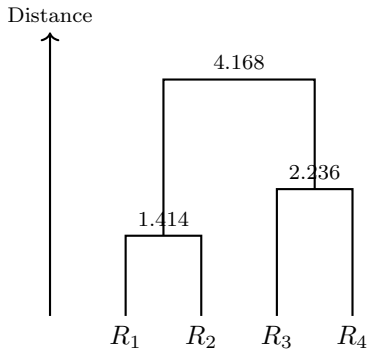
**Step 1:** Smallest distance is $d(R_1, R_2) \approx 1.414$. Merge into $\{R_1, R_2\}$.

$$d(\{R_1, R_2\}, R_3) = \frac{4 + 3.162}{2} = 3.581$$
$$d(\{R_1, R_2\}, R_4) = \frac{5.385 + 4.123}{2} = 4.754$$
$$d(R_3, R_4) = 2.236$$

**Step 2:** Smallest distance is now $d(R_3, R_4) \approx 2.236$. Merge into $\{R_3, R_4\}$.

$$d(\{R_1, R_2\}, \{R_3, R_4\}) = \frac{4 + 5.385 + 3.162 + 4.123}{4} = \frac{16.670}{4} \approx 4.168$$

**Step 3:** Only two clusters remain; merge them at height $\approx 4.168$.




Merge order: $(R_1, R_2)$ at 1.414, $(R_3, R_4)$ at 2.236, root at 4.168

**Reading a Dendrogram and Comparing Linkages:**

1. The **height** of a merge shows how far apart the two clusters were when they joined; lower merges happened first
2. Cutting the dendrogram at any height gives a clustering: cut low for many small clusters, cut high for fewer large clusters
3. **Single linkage** can produce long chained clusters since it only needs one close pair
4. **Complete linkage** produces tighter, more compact clusters since it is controlled by the farthest pair
5. **Average linkage** balances between the two, using the mean of all cross-cluster distances

# k-Means Partitional Clustering

## The k-Means Algorithm

**k-Means Clustering**

k-Means is a **partitional** clustering algorithm that divides $n$ points into exactly $k$ clusters, where $k$ is chosen in advance. Each cluster is represented by its **centroid** (the mean position of all points currently in that cluster), and every point is assigned to the cluster whose centroid is nearest to it.

**Properties of k-Means**

1. k-Means minimizes the **within-cluster sum of squared distances** between points and their cluster centroid *(objective)*
2. The number of clusters $k$ must be chosen before running the algorithm *(fixed $k$)*
3. k-Means always converges (the assignments eventually stop changing), but only to a **local** optimum, not necessarily the best possible clustering *(local convergence)*
4. The final clustering depends on the **initial choice of centroids**; different starting points can give different results *(initialization sensitivity)*

**k-Means Algorithm (Lloyd's Algorithm)**

**Input:** a set of $n$ points, the number of clusters $k$, initial centroids $c_1, \ldots, c_k$
**Output:** $k$ clusters with their final centroids

1. Choose $k$ initial centroids (often $k$ of the data points themselves).
2. Assign every point to the cluster whose centroid is nearest, using Euclidean distance.
3. Recompute each centroid as the mean of all points currently assigned to that cluster.
4. If any point changed cluster in step 2, or any centroid moved in step 3, go back to step 2.
5. Otherwise (no reassignment and no centroid movement) stop; the clustering has converged.
6. Return the final clusters and centroids.

**Example 3.13 — k-Means with Well-Separated Initial Centroids**

Cluster the points $A(1,1)$, $B(2,1)$, $C(4,3)$, $D(5,4)$ using $k = 2$ with initial centroids $c_1 = (1,1)$ and $c_2 = (5,4)$.

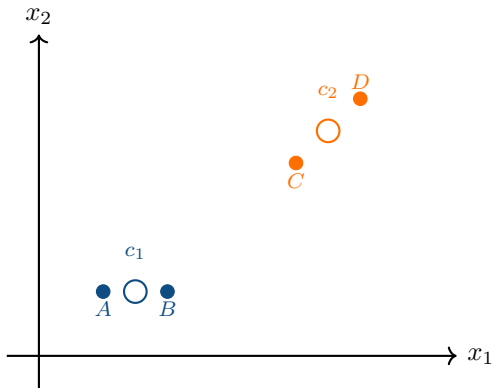
**Solution: Iteration 1 — Assign:**

$$\begin{aligned} d(A,c_1) = 0,\ d(A,c_2) = 5 &\Rightarrow A \to c_1 \\ d(B,c_1) = 1,\ d(B,c_2) \approx 4.243 &\Rightarrow B \to c_1 \\ d(C,c_1) \approx 3.606,\ d(C,c_2) \approx 1.414 &\Rightarrow C \to c_2 \\ d(D,c_1) = 5,\ d(D,c_2) = 0 &\Rightarrow D \to c_2 \end{aligned}$$

**Update:**

$$c_1 = \left(\frac{1+2}{2}, \frac{1+1}{2}\right) = (1.5, 1), \qquad c_2 = \left(\frac{4+5}{2}, \frac{3+4}{2}\right) = (4.5, 3.5)$$

**Iteration 2 — Assign:** Checking distances to the new centroids, $A$ and $B$ remain closest to $c_1$, and $C$ and $D$ remain closest to $c_2$. No point changes cluster, so the centroids do not move again. Converged: $\{A, B\} \to (1.5, 1),\ \{C, D\} \to (4.5, 3.5)$




**Example 3.14 — k-Means with Poorly Chosen Initial Centroids**

Cluster the points $A(1,1)$, $B(2,2)$, $C(3,3)$, $D(8,8)$, $E(9,9)$ using $k = 2$ with initial centroids $c_1 = (1,1)$ and $c_2 = (2,2)$.

**Solution: Iteration 1 — Assign:**

$$A \to c_1\ (d = 0 < 1.414), \quad B \to c_2\ (d = 0 < 1.414), \quad C \to c_2\ (1.414 < 2.828)$$
$$D \to c_2\ (8.485 < 9.899), \quad E \to c_2\ (9.899 < 11.314)$$

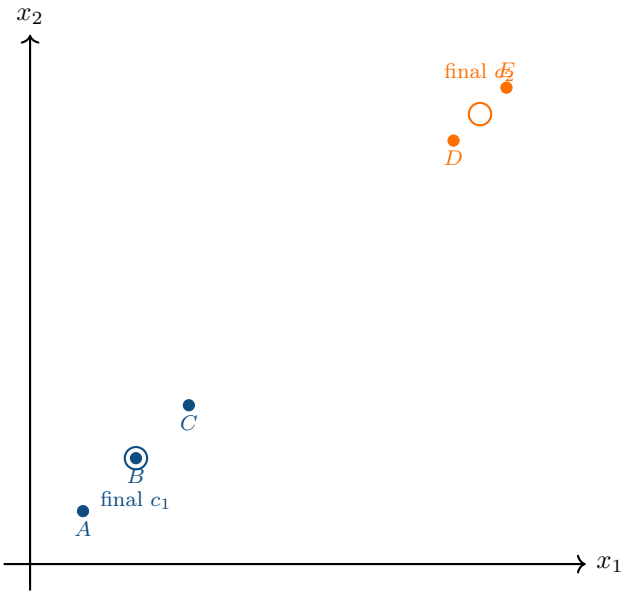
**Update:** $c_1 = (1,1)$ (only $A$); $c_2 = \left(\frac{2+3+8+9}{4}, \frac{2+3+8+9}{4}\right) = (5.5, 5.5)$

**Iteration 2 — Assign (using new centroids):**

$$A \to c_1\ (0 < 6.364), \quad B \to c_1\ (1.414 < 4.950)\ \textbf{[flipped]}, \quad C \to c_1\ (2.828 < 3.536)\ \textbf{[flipped]}$$
$$D \to c_2\ (9.899 > 3.536), \quad E \to c_2\ (11.314 > 4.950)$$

**Update:** $c_1 = \left(\frac{1+2+3}{3}, \frac{1+2+3}{3}\right) = (2,2)$; $c_2 = (8.5, 8.5)$

**Iteration 3 — Assign (using new centroids):** $A, B, C$ all remain closest to $c_1$; $D, E$ remain closest to $c_2$. No further changes. Converged: $\{A, B, C\} \to (2,2),\ \{D, E\} \to (8.5, 8.5)$



Notice how the poor initial choice ($c_1, c_2$ starting right next to each other) caused $B$ and $C$ to be misassigned in Iteration 1 and only get corrected in later iterations — this is exactly the initialization sensitivity described in the theorembox above. ✓

**Example 3.15 — k-Means with Three Clusters**

Cluster the points $A(1,1)$, $B(1,2)$, $C(8,1)$, $D(8,2)$, $E(4,8)$, $F(5,9)$ using $k = 3$ with initial centroids $c_1 = (1,1)$, $c_2 = (8,1)$, $c_3 = (4,8)$.

**Solution: Iteration 1 — Assign:**

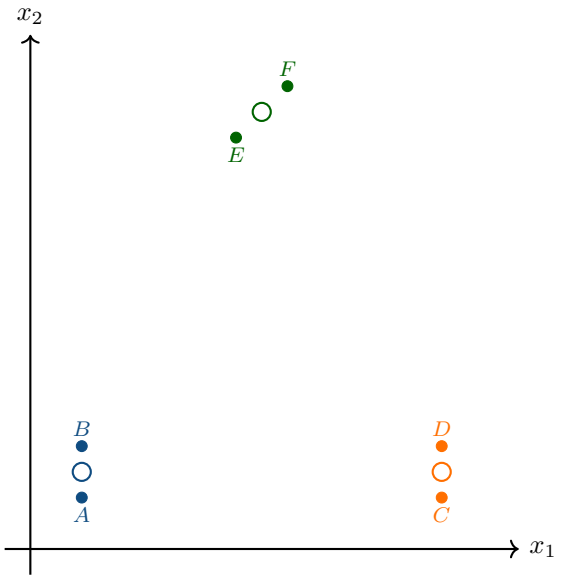
$$A \to c_1\ (0), \quad B \to c_1\ (1 < 7.071, 6.708), \quad C \to c_2\ (0), \quad D \to c_2\ (1 < 7.211)$$
$$E \to c_3\ (0), \quad F \to c_3\ (1.414 < 8.944, 8.544)$$

**Update:**

$$c_1 = (1, 1.5), \qquad c_2 = (8, 1.5), \qquad c_3 = (4.5, 8.5)$$

**Iteration 2 — Assign (using new centroids):** Every point remains closest to its own cluster's new centroid (each point is only 0.5 or 0.707 from its own centroid, while the nearest other centroid is several units away). No further changes.

Converged: $\{A, B\} \to (1, 1.5)$, $\{C, D\} \to (8, 1.5)$, $\{E, F\} \to (4.5, 8.5)$




**Practical Notes on k-Means:**

1. At every iteration, check **both** whether any point changed cluster **and** whether centroids moved — stop only when neither happens
2. Poorly chosen initial centroids (Example 3.14) can cause extra iterations or a worse final clustering; k-Means is often run several times with different random starts and the best result kept
3. Unlike Decision Trees or Naive Bayes, k-Means never uses class labels — the "correct" grouping is discovered purely from distances

## Sum of Squared Errors and Choosing k

**Sum of Squared Errors (SSE)**

The Sum of Squared Errors measures how tightly the points fit their assigned clusters:

$$SSE = \sum_{j=1}^{k} \sum_{x \in \text{Cluster}_j} d(x, c_j)^2$$

where $c_j$ is the centroid of Cluster $j$. A **lower** SSE means the points are, on average, closer to their own cluster's centroid — a tighter, better clustering.

This is simply the squared-distance idea from Section 3.1.2, summed over every point in every cluster.

**Example 3.16 — SSE of the Clustering from Example 3.13**

Using the final clusters from Example 3.13 — $\{A(1,1), B(2,1)\} \to c_1 = (1.5, 1)$ and $\{C(4,3), D(5,4)\} \to c_2 = (4.5, 3.5)$ — compute the total SSE.

**Solution:**

$$d(A, c_1)^2 = (1 - 1.5)^2 + (1 - 1)^2 = 0.25$$
$$d(B, c_1)^2 = (2 - 1.5)^2 + (1 - 1)^2 = 0.25$$
$$d(C, c_2)^2 = (4 - 4.5)^2 + (3 - 3.5)^2 = 0.5$$
$$d(D, c_2)^2 = (5 - 4.5)^2 + (4 - 3.5)^2 = 0.5$$
$$SSE = 0.25 + 0.25 + 0.5 + 0.5$$

$\boxed{SSE = 1.5}$

**Example 3.17 — SSE of a Single Cluster**

A cluster contains the points $(0,0)$, $(2,0)$, $(1,3)$. Compute the centroid and the SSE for this cluster.

**Solution:**

$$c = \left(\frac{0+2+1}{3}, \frac{0+0+3}{3}\right) = (1, 1)$$
$$d((0,0), c)^2 = 1 + 1 = 2$$
$$d((2,0), c)^2 = 1 + 1 = 2$$
$$d((1,3), c)^2 = 0 + 4 = 4$$
$$SSE = 2 + 2 + 4$$

$\boxed{SSE = 8}$

**Example 3.18 — Comparing SSE for k**

For the points $(1,1)$, $(1,2)$, $(5,5)$, $(5,6)$, compute the SSE when using $k = 1$ (one cluster containing all points) and compare it to the SSE when using $k = 2$ (clusters $\{(1,1), (1,2)\}$ and $\{(5,5), (5,6)\}$).

**Solution: Case $k = 1$:** centroid $= \left(\frac{1+1+5+5}{4}, \frac{1+2+5+6}{4}\right) = (3, 3.5)$

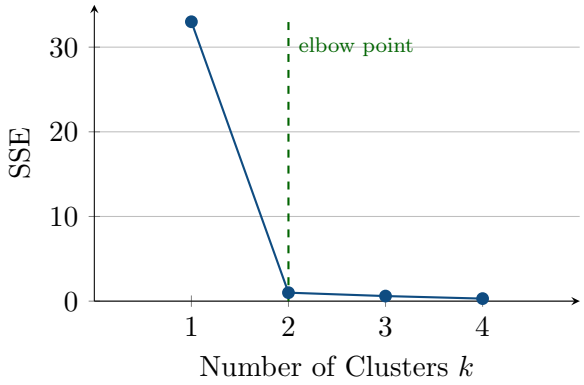
$$SSE_{k=1} = [(1-3)^2 + (1-3.5)^2] + [(1-3)^2 + (2-3.5)^2] + [(5-3)^2 + (5-3.5)^2] + [(5-3)^2 + (6-3.5)^2]$$
$$= [4 + 6.25] + [4 + 2.25] + [4 + 2.25] + [4 + 6.25]$$

$\boxed{SSE_{k=1} = 33}$

**Case $k = 2$:** centroids $(1, 1.5)$ and $(5, 5.5)$

$$SSE_{k=2} = [(1-1)^2 + (1-1.5)^2] + [(1-1)^2 + (2-1.5)^2] + [(5-5)^2 + (5-5.5)^2] + [(5-5)^2 + (6-5.5)^2]$$
$$= 0.25 + 0.25 + 0.25 + 0.25$$

$\boxed{SSE_{k=2} = 1}$ Increasing $k$ from 1 to 2 dropped the SSE sharply, from 33 down to 1, because each point now sits much closer to its own (smaller) cluster's centroid.




**The Elbow Method for Choosing k:**

1. SSE always **decreases** as $k$ increases (more clusters means points sit closer to their centroids), reaching $SSE = 0$ when $k = n$
2. Plot SSE against $k$: the curve drops sharply at first, then flattens out
3. The **elbow point** — where the drop suddenly slows down, as at $k = 2$ in Example 3.18 — is chosen as a good balance between simplicity (few clusters) and tightness (low SSE)
4. Choosing $k$ too large just to minimize SSE defeats the purpose of clustering, since $k = n$ trivially gives $SSE = 0$ with every point in its own cluster

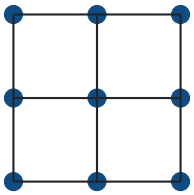
# Self-Organizing Maps (SOM)

## Introduction to Self-Organizing Maps

**Self-Organizing Map (SOM)**

A Self-Organizing Map is an unsupervised neural network that arranges a grid of **neurons** (each holding a weight vector of the same dimension as the input) so that neurons which are close together on the grid end up responding to similar input patterns. Unlike k-Means, which produces unordered clusters, a SOM preserves **topology** — nearby neurons on the map represent nearby regions of the input space.

This is similar in spirit to k-Means (each neuron's weight vector acts like a centroid), except the neurons also influence their grid-neighbours during training, which keeps similar patterns mapped near each other.



$3 \times 3$ grid of neurons, each with a weight vector

## Best Matching Unit and Weight Update

**Best Matching Unit (BMU)**

For a given input vector $x$, the Best Matching Unit is the neuron whose weight vector $w_j$ is closest to $x$ (using Euclidean distance):

$$\text{BMU} = \arg\min_j \; d(x, w_j)$$

**SOM Weight Update Rule**

After finding the BMU for input $x$, every neuron $j$ within the BMU's neighbourhood updates its weight vector to move slightly closer to $x$:

$$w_j^{\text{new}} = w_j^{\text{old}} + \eta \cdot \left(x - w_j^{\text{old}}\right)$$

where $\eta$ is the learning rate ($0 < \eta \leq 1$). Neurons outside the neighbourhood are left unchanged.

This update rule is just "move a fraction $\eta$ of the way from the old weight toward the input" — the same kind of small-step adjustment used in gradient-based learning.

**Self-Organizing Map Training (Simplified)**

**Input:** training points, a grid of neurons with initial weight vectors, learning rate $\eta$, neighbourhood radius $r$

**Output:** trained neuron weight vectors

1. Pick a training input $x$.
2. Compute the distance from $x$ to every neuron's weight vector and find the BMU (smallest distance).

3. Identify all neurons within radius $r$ of the BMU on the grid (the neighbourhood).
4. Update the weight vector of the BMU and every neuron in its neighbourhood using $w_j^{\text{new}} = w_j^{\text{old}} + \eta(x - w_j^{\text{old}})$.
5. Repeat steps 1 to 4 for every training input (one full pass is called an epoch).
6. Optionally shrink $\eta$ and $r$ slightly and repeat for more epochs.
7. Return the final weight vectors.

**Example 3.19 — Finding the Best Matching Unit**

Three neurons have weight vectors $w_1 = (1, 1)$, $w_2 = (4, 4)$, $w_3 = (8, 1)$. For input $x = (5, 3)$, find the BMU.

**Solution:**

$$d(x, w_1) = \sqrt{(5-1)^2 + (3-1)^2} = \sqrt{16+4} = \sqrt{20} \approx 4.472$$
$$d(x, w_2) = \sqrt{(5-4)^2 + (3-4)^2} = \sqrt{1+1} = \sqrt{2} \approx 1.414$$
$$d(x, w_3) = \sqrt{(5-8)^2 + (3-1)^2} = \sqrt{9+4} = \sqrt{13} \approx 3.606$$

The smallest distance is $d(x, w_2) \approx 1.414$. $\boxed{\text{BMU} = w_2}$

**Example 3.20 — Updating the BMU's Weight Vector**

Using the BMU $w_2 = (4, 4)$ found in Example 3.19 and input $x = (5, 3)$, update $w_2$ with learning rate $\eta = 0.5$.

**Solution:**

$$\begin{aligned} w_2^{\text{new}} &= w_2^{\text{old}} + \eta(x - w_2^{\text{old}}) \\ &= (4, 4) + 0.5\big[(5, 3) - (4, 4)\big] \\ &= (4, 4) + 0.5(1, -1) \\ &= (4, 4) + (0.5, -0.5) \end{aligned}$$

$\boxed{w_2^{\text{new}} = (4.5, 3.5)}$

**Example 3.21 — Updating a Neighbour Neuron**

Using the same setup as Example 3.20, suppose neuron $w_3 = (8, 1)$ is also inside the BMU's neighbourhood. Update $w_3$ with the same input $x = (5, 3)$ and learning rate $\eta = 0.5$.

**Solution:**

$$\begin{aligned} w_3^{\text{new}} &= w_3^{\text{old}} + \eta(x - w_3^{\text{old}}) \\ &= (8, 1) + 0.5\big[(5, 3) - (8, 1)\big] \\ &= (8, 1) + 0.5(-3, 2) \\ &= (8, 1) + (-1.5, 1) \end{aligned}$$

$\boxed{w_3^{\text{new}} = (6.5, 2)}$ Notice that $w_3$ still moves toward $x$, but starts farther away and so remains farther from $x$ than the BMU does — this is expected since it is only a neighbour, not the BMU itself.

**SOM vs k-Means:**

1. Both find a "closest representative vector" for each input, exactly like a BMU (SOM) or nearest centroid (k-Means)
2. k-Means updates **only** the winning centroid; SOM updates the winning neuron **and** its grid neighbours, which is what creates the topology-preserving map
3. In SOM, neurons that are close on the grid end up with similar weight vectors after training, so the map can be visualized to show which regions of input space are related
4. A larger learning rate $\eta$ makes each update move the weight vector further toward the input, exactly as seen by comparing the update sizes in Examples 3.20 and 3.21

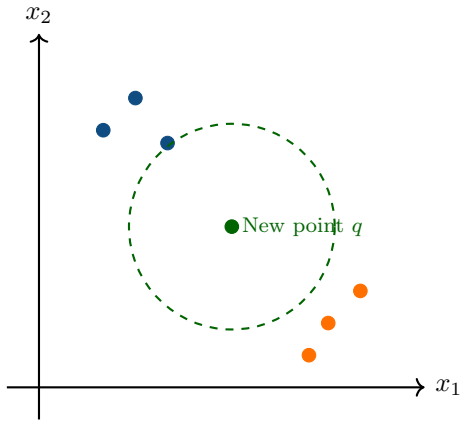
# k-Nearest Neighbor Algorithm

## The k-NN Idea

**k-Nearest Neighbor (k-NN)**

k-Nearest Neighbor is an **instance-based** (lazy) learning algorithm. It stores all the training examples and performs no explicit training phase. To classify a new point, it finds the $k$ training points closest to it (using a distance measure such as Euclidean distance) and assigns the **majority class** among those $k$ neighbors.

Unlike Decision Trees or Naive Bayes, which build a model in advance, k-NN waits until a new point actually needs to be classified — this is why it is called a *lazy* learner.




## The k-NN Algorithm

**k-Nearest Neighbor Algorithm**

**Input:** training points with known class labels, a test point $q$, the number of neighbors $k$

**Output:** predicted class label for $q$

1. Compute the distance from $q$ to every training point.
2. Sort the training points by distance to $q$ in increasing order.
3. Select the $k$ closest training points (the $k$ nearest neighbors).
4. Count how many of these $k$ neighbors belong to each class.
5. Assign $q$ the class that appears most often among the $k$ neighbors (majority vote).
6. Return the predicted class.

**Example 3.22 — Classifying with k**

| Point | $x_1$ | $x_2$ | Class |
|---|---|---|---|
| $P_1$ | 1 | 1 | A |
| $P_2$ | 2 | 1 | A |
| $P_3$ | 1 | 2 | A |
| $P_4$ | 6 | 6 | B |
| $P_5$ | 7 | 6 | B |

Classify the query point $q = (2, 2)$ using k-NN with $k = 3$.

**Solution:**

$$d(q, P_1) = \sqrt{1+1} = \sqrt{2} \approx 1.414$$
$$d(q, P_2) = \sqrt{0+1} = 1$$
$$d(q, P_3) = \sqrt{1+0} = 1$$
$$d(q, P_4) = \sqrt{16+16} \approx 5.657$$
$$d(q, P_5) = \sqrt{25+16} \approx 6.403$$

Sorted by distance: $P_2(1), P_3(1), P_1(1.414), P_4(5.657), P_5(6.403)$. The 3 nearest neighbors are $P_2, P_3, P_1$ — all class A. $\boxed{\text{Predicted class} = A}$

**Example 3.23 — Classifying with k**

Using the same dataset as Example 3.22, classify $q = (6, 5)$ using k-NN with $k = 1$.

**Solution:**

$$d(q, P_1) = \sqrt{25+16} \approx 6.403$$
$$d(q, P_2) = \sqrt{16+16} \approx 5.657$$
$$d(q, P_3) = \sqrt{25+9} \approx 5.831$$
$$d(q, P_4) = \sqrt{0+1} = 1$$
$$d(q, P_5) = \sqrt{1+1} \approx 1.414$$

The single nearest neighbor is $P_4$ (distance 1), which is class B. $\boxed{\text{Predicted class} = B}$

**Example 3.24 — Classifying with a Tie Broken by Closer Points**

| Point | $x_1$ | $x_2$ | Class |
|---|---|---|---|
| $P_1$ | 1 | 1 | A |
| $P_2$ | 2 | 2 | A |
| $P_3$ | 5 | 1 | B |
| $P_4$ | 6 | 2 | B |

Classify the query point $q = (3.5, 1.5)$ using k-NN with $k = 4$.

**Solution:**

$$d(q, P_1) = \sqrt{6.25+0.25} = \sqrt{6.5} \approx 2.550$$
$$d(q, P_2) = \sqrt{2.25+0.25} = \sqrt{2.5} \approx 1.581$$
$$d(q, P_3) = \sqrt{2.25+0.25} = \sqrt{2.5} \approx 1.581$$
$$d(q, P_4) = \sqrt{6.25+0.25} = \sqrt{6.5} \approx 2.550$$

With $k = 4$, all four points are neighbors: 2 of class A ($P_1, P_2$) and 2 of class B ($P_3, P_4$) — an exact tie. $\boxed{\text{Tie between A and B: an odd } k \text{ (e.g. } k = 3) \text{ should be used instead to avoid this}}$ This illustrates exactly why $k$ is usually chosen to be odd for two-class problems, so majority voting cannot tie. ✓

**Choosing k in k-NN:**

1. A **small** $k$ (e.g. $k = 1$) is sensitive to noise — one mislabeled nearby point can flip the

prediction

2. A **large** $k$ smooths out noise but risks pulling in points from a different, more distant class
3. $k$ is usually chosen **odd** for two-class problems specifically to avoid ties, as seen in Example 3.24
4. k-NN needs no explicit training phase, but classifying each new point requires computing its distance to **every** training point, which is why it is called "lazy" learning

**Example 3.25 — Effect of Increasing k**

| Point | $x_1$ | $x_2$ | Class |
|---|---|---|---|
| $P_1$ | 1 | 1 | A |
| $P_2$ | 2 | 1 | A |
| $P_3$ | 2 | 3 | B |
| $P_4$ | 8 | 8 | B |
| $P_5$ | 9 | 9 | B |

Classify $q = (2, 2)$ using k-NN with (a) $k = 1$ and (b) $k = 3$, and compare the results.

**Solution:**

$$d(q, P_1) = \sqrt{1+1} \approx 1.414$$
$$d(q, P_2) = \sqrt{0+1} = 1$$
$$d(q, P_3) = \sqrt{0+1} = 1$$
$$d(q, P_4) = \sqrt{36+36} \approx 8.485$$
$$d(q, P_5) = \sqrt{49+49} \approx 9.899$$

**(a)** $k = 1$**:** There is a tie for nearest between $P_2$ (distance 1, class A) and $P_3$ (distance 1, class B); assuming $P_2$ is selected first, the prediction is A. **(b)** $k = 3$**:** The 3 nearest neighbors are $P_2(A)$, $P_3(B)$, $P_1(A)$ — majority is 2 A's vs 1 B. $\boxed{k = 1 \Rightarrow A \text{ (tie-sensitive)}, \qquad k = 3 \Rightarrow A \text{ (clear majority)}}$ Here both choices happen to agree, but $k = 3$ reaches its answer through a clear majority vote rather than a distance tie, making it the more robust choice.

## Practice Problems

1. Compute the Euclidean distance between the points $(3, 4)$ and $(7, 1)$.

2. Cluster $A = \{(1, 1), (2, 3)\}$ and Cluster $B = \{(5, 1), (6, 4)\}$. Compute the single linkage distance $d(A, B)$.

3. Using the same clusters as the previous question, compute the complete linkage distance $d(A, B)$.

4. Using the same clusters as the previous two questions, compute the average linkage distance $d(A, B)$.

5. Cluster the points $P_1 = (1, 1)$, $P_2 = (2, 2)$, $P_3 = (9, 1)$, $P_4 = (10, 2)$ using hierarchical agglomerative clustering with single linkage. Show every merge step and draw the resulting dendrogram.

6. Cluster the points $Q_1 = (0, 0)$, $Q_2 = (1, 0)$, $Q_3 = (0, 5)$, $Q_4 = (1, 5)$ using hierarchical agglomerative clustering with complete linkage. Show every merge step and draw the resulting dendrogram.

7. Using k-Means with $k = 2$ and initial centroids $c_1 = (0, 0)$ and $c_2 = (10, 10)$, perform one full iteration (assign and update) for the points $(1, 1)$, $(2, 1)$, $(9, 9)$, $(8, 10)$.

8. For the clusters obtained in the previous question, compute the total Sum of Squared Errors (SSE).

9. Explain, using a small example of your own with at least 4 points, why choosing $k$ too large defeats the purpose of the SSE-based elbow method.

10. Three neurons in a Self-Organizing Map have weight vectors $w_1 = (2, 2)$, $w_2 = (6, 2)$, $w_3 = (4, 8)$. For input $x = (5, 3)$, find the Best Matching Unit (BMU).

11. Using the BMU found in the previous question, update its weight vector with learning rate $\eta = 0.4$.

12. If a neighboring neuron has weight vector $w = (4, 8)$ and the same input $x = (5, 3)$ is applied with learning rate $\eta = 0.2$, compute its updated weight vector.

13. Given the training points $(1, 1, A)$, $(1, 2, A)$, $(2, 1, A)$, $(8, 8, B)$, $(9, 8, B)$ (format: $x_1, x_2$, Class), classify the query point $q = (2, 2)$ using k-NN with $k = 3$.

14. Using the same dataset as the previous question, classify $q = (8, 7)$ using k-NN with $k = 1$.

15. Explain, using a short example, why an odd value of $k$ is generally preferred over an even value of $k$ in k-NN for a two-class classification problem.

# Chapter 4

# Probabilistic and Reinforcement Learning

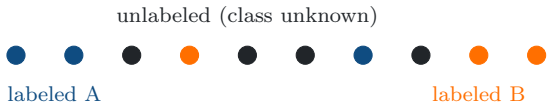
## Semi-Supervised Learning with the EM Algorithm

### Labeled and Unlabeled Data

**Semi-Supervised Learning**

Semi-supervised learning trains a model using a small **labeled** set $L = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ together with a much larger **unlabeled** set $U = \{x_{m+1}, \ldots, x_{m+n}\}$ whose true class $y$ is unknown. The goal is to use the pattern found in both sets together to build a better classifier than the labeled data alone could produce.

This sits exactly between the supervised setting of Chapter 2 (every example labeled) and the unsupervised setting of Chapter 3 (no example labeled).




**Example 4.1 — Recognizing a Semi-Supervised Dataset**

A dataset has 100 emails: 15 are marked Spam or NotSpam by a human, and the remaining 85 have no label at all. Is this a semi-supervised learning problem?

**Solution:** Labeled count = 15, Unlabeled count = 85. Since both a nonzero labeled set **and** a nonzero unlabeled set exist together in the same training data:
Yes, this is a semi-supervised learning problem

**Example 4.2 — Recognizing a Fully Supervised Dataset**

A dataset has 50 patient records, and every single record has a confirmed Disease/NoDisease label. Is this a semi-supervised learning problem?

**Solution:** Labeled count = 50, Unlabeled count = 0. Since there is no unlabeled portion at all: No, this is an ordinary (fully) supervised learning problem

**Example 4.3 — Recognizing an Unsupervised Dataset**

A dataset has 200 customer purchase records, and none of them have any class label attached. Is this a semi-supervised learning problem?

**Solution:** Labeled count = 0, Unlabeled count = 200. Since there is no labeled portion at all to anchor the classes: No, this is an unsupervised learning problem (clustering, Chapter 3)

**Where Semi-Supervised Learning Fits:**

1. **Supervised** (Chapter 2): $L$ only, $U = \varnothing$
2. **Unsupervised** (Chapter 3): $U$ only, $L = \varnothing$
3. **Semi-supervised** (this section): both $L \neq \varnothing$ and $U \neq \varnothing$ used together
4. Semi-supervised learning is valuable because labeling data (by a human) is expensive, while unlabeled data is often cheap and plentiful

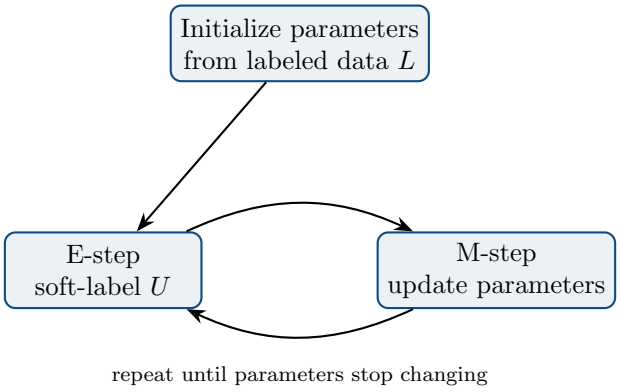
## The Expectation-Maximization Framework

**Expectation-Maximization (EM) Algorithm**

EM is an iterative procedure for estimating model parameters when some data is incomplete (here, the missing piece is the class label $y$ of the unlabeled examples). It alternates between two steps until the parameters stop changing:

- **E-step (Expectation):** using the current parameters, compute the probability of each possible class for every unlabeled example — a **soft** (fractional) label rather than a hard 0/1 label.
- **M-step (Maximization):** using the labeled data together with these fractional soft labels, recompute the parameters that best fit all the data.

This is the same "guess, then refine, then guess again" idea as k-Means (Section 3.2): k-Means alternates Assign and Update; EM alternates E-step and M-step, except EM assigns *fractional* membership instead of putting each point in exactly one cluster.




**Properties of the EM Algorithm**

1. Each round of E-step followed by M-step never decreases how well the parameters fit the data *(monotonic improvement)*
2. EM is guaranteed to converge, but only to a **local** optimum, not necessarily the best possible parameters *(local convergence, same caution as k-Means)*
3. The E-step produces **soft** (fractional, e.g. 0.3/0.7) class probabilities for unlabeled points, not hard single-class labels *(soft assignment)*

4. The initial parameters (estimated from $L$ alone) affect where EM converges *(initialization sensitivity)*

### EM for Semi-Supervised Naive Bayes Classification

**EM Algorithm for Semi-Supervised Naive Bayes**

**Input:** labeled examples $L$ with known class and attribute values, unlabeled examples $U$ with only attribute values

**Output:** class priors $P(C_k)$, attribute likelihoods $P(A_i \mid C_k)$, and soft labels for $U$

1. Using only $L$, estimate initial priors $P(C_k)$ and likelihoods $P(A_i \mid C_k)$ exactly as in ordinary Naive Bayes (Section 2.2).
2. **E-step:** for every unlabeled example $x \in U$, compute the (unnormalized) score $P(C_k)\prod_i P(A_i \mid C_k)$ for each class $C_k$, then normalize these scores to sum to 1 — this gives the soft label $P(C_k \mid x)$.
3. **M-step:** recompute every prior and likelihood using both the labeled data (weight 1 per example) and every unlabeled example (weighted by its soft label from step 2) as fractional counts.
4. If the parameters changed from the previous round, go back to step 2; otherwise stop.
5. Return the final parameters and the final soft labels for $U$.

**Example 4.4 — Initializing Parameters from Labeled Data**

The labeled set $L$ below records Weather and whether cricket was played.

| Weather | PlayCricket |
|---|---|
| Sunny | Yes |
| Sunny | Yes |
| Rainy | Yes |
| Rainy | No |
| Rainy | No |
| Sunny | No |

Compute the initial Naive Bayes parameters $P(\text{Yes})$, $P(\text{No})$, $P(\text{Sunny} \mid \text{Yes})$, $P(\text{Rainy} \mid \text{Yes})$, $P(\text{Sunny} \mid \text{No})$, $P(\text{Rainy} \mid \text{No})$.

**Solution:**

$$P(\text{Yes}) = \frac{3}{6} = 0.5, \qquad P(\text{No}) = \frac{3}{6} = 0.5$$

$$P(\text{Sunny} \mid \text{Yes}) = \frac{2}{3} \approx 0.667, \qquad P(\text{Rainy} \mid \text{Yes}) = \frac{1}{3} \approx 0.333$$

$$P(\text{Sunny} \mid \text{No}) = \frac{1}{3} \approx 0.333, \qquad P(\text{Rainy} \mid \text{No}) = \frac{2}{3} \approx 0.667$$

Initial parameters computed exactly as ordinary Naive Bayes on $L$ alone

**Example 4.5 — E-step: Soft-Labeling an Unlabeled Example**

An unlabeled example $U_1$ has Weather = Rainy. Using the parameters from Example 4.4, compute the soft label $P(\text{Yes} \mid U_1)$ and $P(\text{No} \mid U_1)$.

**Solution:**

$$\text{score}(\text{Yes}) = P(\text{Yes}) \cdot P(\text{Rainy} \mid \text{Yes}) = (0.5)(0.333) \approx 0.1667$$
$$\text{score}(\text{No}) = P(\text{No}) \cdot P(\text{Rainy} \mid \text{No}) = (0.5)(0.667) \approx 0.3333$$
$$\text{Total} = 0.1667 + 0.3333 = 0.5$$
$$P(\text{Yes} \mid U_1) = \frac{0.1667}{0.5} \approx 0.333, \qquad P(\text{No} \mid U_1) = \frac{0.3333}{0.5} \approx 0.667$$

$U_1$ is soft-labeled 33.3% Yes and 66.7% No (not a hard 0/1 label)

**Example 4.6 — M-step: Updating Parameters with Fractional Counts**

Using the labeled set $L$ from Example 4.4 and the soft-labeled $U_1$ (Weather = Rainy) from Example 4.5, recompute $P(\text{Yes})$, $P(\text{No})$, $P(\text{Rainy} \mid \text{Yes})$, and $P(\text{Rainy} \mid \text{No})$.

**Solution:** $U_1$ contributes a fractional count of 0.333 toward the Yes class and 0.667 toward the No class.

$$\text{New Yes count} = 3 + 0.333 = 3.333, \qquad \text{New No count} = 3 + 0.667 = 3.667$$
$$\text{New total} = 3.333 + 3.667 = 7$$
$$P(\text{Yes}) = \frac{3.333}{7} \approx 0.476, \qquad P(\text{No}) = \frac{3.667}{7} \approx 0.524$$

For the attribute likelihood, $U_1$'s Rainy value adds 0.333 to the Yes-Rainy count and 0.667 to the No-Rainy count:

$$\text{New Yes-Rainy count} = 1 + 0.333 = 1.333, \qquad \text{New Yes total} = 3.333$$
$$P(\text{Rainy} \mid \text{Yes}) = \frac{1.333}{3.333} = 0.4$$
$$\text{New No-Rainy count} = 2 + 0.667 = 2.667, \qquad \text{New No total} = 3.667$$
$$P(\text{Rainy} \mid \text{No}) = \frac{2.667}{3.667} \approx 0.727$$

$P(\text{Yes}) \approx 0.476,\ P(\text{No}) \approx 0.524,\ P(\text{Rainy} \mid \text{Yes}) = 0.4,\ P(\text{Rainy} \mid \text{No}) \approx 0.727$ These updated parameters would now be used to redo the E-step, and the cycle repeats until the parameters stop changing (convergence).

**Why EM Helps Here:**

1. Using $L$ alone (Example 4.4) gives parameters based on only 6 examples
2. EM lets the unlabeled example $U_1$ contribute information too, **proportional to how confident** the model currently is about its class (Example 4.5), rather than guessing a hard label
3. After the M-step (Example 4.6), the parameters shift slightly — toward whichever class the soft label favored more (here, No shifted up because $U_1$ was 66.7% likely No)
4. With a large unlabeled set $U$, this fractional-counting process can noticeably sharpen a classifier trained from only a small labeled set $L$

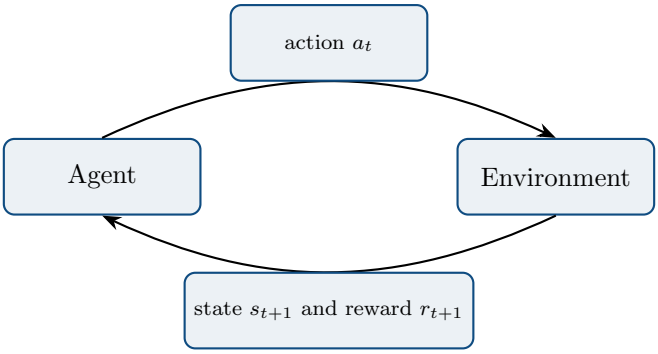
# Reinforcement Learning and Hidden Markov Models

## Introduction to Reinforcement Learning

**Reinforcement Learning**

Reinforcement Learning (RL) is a learning setting in which an **agent** interacts with an **environment** over a sequence of time steps. At each step the agent observes the current **state**, chooses an **action**, and receives a numerical **reward** along with the next state. The agent learns, purely through this trial-and-error interaction, a **policy** (a rule for choosing actions) that maximizes the total reward it collects over time.

Unlike every algorithm in Chapters 2 and 3, RL has no fixed dataset at all — the agent generates its own training data by acting and observing the consequences.




**Core RL Terminology:**

1. **Agent** the learner or decision maker
2. **Environment** everything the agent interacts with and that responds to its actions
3. **State** $s_t$ a description of the situation at time $t$
4. **Action** $a_t$ a choice the agent makes at time $t$
5. **Reward** $r_t$ a numerical signal received after taking an action, indicating how good or bad the outcome was
6. **Policy** the agent's strategy for choosing an action given a state

**Discounted Return**

The return $G_t$ is the total reward the agent collects from time $t$ onward, with rewards further in the future counted for less using a **discount factor** $\gamma$ where $0 \leq \gamma < 1$:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

This is the same geometric-series idea used in compound interest, just applied to rewards shrinking into the future instead of money growing.

**Computing Discounted Return**

**Input:** a sequence of rewards $r_1, r_2, \ldots, r_n$ received after time $t$, discount factor $\gamma$
**Output:** the discounted return $G_t$

1. Multiply the first reward $r_1$ by $\gamma^0 = 1$.

2. Multiply the second reward $r_2$ by $\gamma^1$.
3. Continue multiplying each subsequent reward $r_k$ by $\gamma^{k-1}$.
4. Sum all the discounted rewards to obtain $G_t$.

**Example 4.7 — Discounted Return with a High Discount Factor**

An agent receives rewards $4, 3, 2$ over three consecutive steps. Compute $G_0$ using $\gamma = 0.9$.

**Solution:**

$$\begin{aligned} G_0 &= 4 + (0.9)(3) + (0.9)^2(2) \\ &= 4 + 2.7 + (0.81)(2) \\ &= 4 + 2.7 + 1.62 \end{aligned}$$

$\boxed{G_0 = 8.32}$

**Example 4.8 — Same Rewards with a Low Discount Factor**

Using the same rewards as Example 4.7 $(4, 3, 2)$, compute $G_0$ using $\gamma = 0.5$.

**Solution:**

$$\begin{aligned} G_0 &= 4 + (0.5)(3) + (0.5)^2(2) \\ &= 4 + 1.5 + (0.25)(2) \\ &= 4 + 1.5 + 0.5 \end{aligned}$$

$\boxed{G_0 = 6}$ Comparing Examples 4.7 and 4.8: the closer $\gamma$ is to 1, the more the later rewards (3 and 2) count toward the total; the closer $\gamma$ is to 0, the more the agent behaves as if only the immediate reward mattered.

**Example 4.9 — Delayed Large Reward**

An agent receives rewards $-1, -1, 10$ over three steps (two small penalties followed by a large reward). Compute $G_0$ using $\gamma = 0.9$.

**Solution:**

$$\begin{aligned} G_0 &= -1 + (0.9)(-1) + (0.9)^2(10) \\ &= -1 - 0.9 + (0.81)(10) \\ &= -1 - 0.9 + 8.1 \end{aligned}$$

$\boxed{G_0 = 6.2}$ Even though the immediate reward is negative, the large discounted future reward makes the overall return positive — this is exactly why an RL agent must plan for delayed consequences rather than just the next single reward.

**Reading the Discount Factor $\gamma$:**

1. $\gamma$ close to 0: the agent is short-sighted and cares mainly about the immediate reward
2. $\gamma$ close to 1: the agent is far-sighted and gives future rewards nearly as much weight as immediate ones
3. $\gamma$ must satisfy $0 \leq \gamma < 1$ so that the infinite sum $G_t$ stays finite for an ongoing task

## Markov Chains

**Markov Chain**

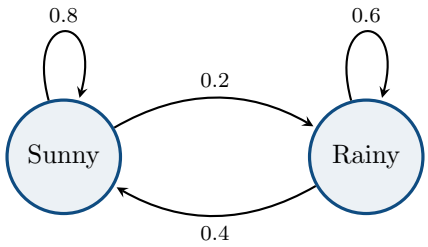
A Markov chain is a system that moves between a fixed set of **states** over time, where the transition to the next state depends only on probabilities attached to the current state. Every possible transition is described by a **transition probability** $P(s_{t+1} = j \mid s_t = i)$.

**Markov Property**

The Markov property states that the future state depends only on the present state, not on the sequence of states that preceded it:

$$P(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_1) = P(s_{t+1} \mid s_t)$$

This is what makes the system "memoryless" beyond the current state.




**Properties of a Transition Matrix**

1. Every transition probability satisfies $0 \leq P(j \mid i) \leq 1$ *(valid probability)*
2. The probabilities leaving any single state must sum to exactly 1 *(row sums to 1)*
3. The probability of an entire path is the **product** of the transition probabilities along that path *(chain rule under the Markov property)*

**Computing the Probability of a State Sequence**

**Input:** a transition probability table, a known starting state, an ordered sequence of future states

**Output:** the probability of observing exactly that sequence of states

1. Start with probability 1 at the given starting state.
2. For each step in the sequence, multiply the running probability by the transition probability from the current state to the next state in the sequence.
3. Move to the next state and repeat step 2 until the sequence ends.
4. Return the final product as the probability of the sequence.

**Example 4.10 — Probability of a Weather Sequence**

Using the transition probabilities in the diagram above (Sunny→Sunny 0.8, Sunny→Rainy 0.2, Rainy→Rainy 0.6, Rainy→Sunny 0.4), find the probability of the sequence Sunny, Sunny, Rainy given that day 1 is known to be Sunny.

**Solution:**

$$\begin{aligned} P(\text{Sunny}, \text{Sunny}, \text{Rainy}) &= P(S_1 = \text{Sunny}) \cdot P(\text{Sunny} \to \text{Sunny}) \cdot P(\text{Sunny} \to \text{Rainy}) \\ &= 1 \times 0.8 \times 0.2 \end{aligned}$$

$\boxed{P = 0.16}$

**Example 4.11 — Probability of a Longer Sequence**

Using the same transition probabilities, find the probability of the sequence Rainy, Rainy, Sunny, Sunny given that day 1 is known to be Rainy.

**Solution:**

$$\begin{aligned} P(\text{Rainy}, \text{Rainy}, \text{Sunny}, \text{Sunny}) &= 1 \times P(\text{Rainy} \to \text{Rainy}) \times P(\text{Rainy} \to \text{Sunny}) \times P(\text{Sunny} \to \text{Sunny}) \\ &= 1 \times 0.6 \times 0.4 \times 0.8 \end{aligned}$$

$\boxed{P = 0.192}$

**Example 4.12 — Two-Step Transition Probability**

Using the same transition probabilities, find the probability that day 3 is Sunny given that day 1 is known to be Sunny (day 2 can be either state).

**Solution:** Since day 2 is unknown, sum over both possibilities for day 2 (law of total probability):

$$\begin{aligned} P(S_3 = \text{Sunny} \mid S_1 = \text{Sunny}) &= P(S \to S)P(S \to S) + P(S \to R)P(R \to S) \\ &= (0.8)(0.8) + (0.2)(0.4) \\ &= 0.64 + 0.08 \end{aligned}$$

$\boxed{P(S_3 = \text{Sunny} \mid S_1 = \text{Sunny}) = 0.72}$ Notice the difference from Example 4.10: here the in-between state is **not** fixed, so every possible path to the destination must be added together, rather than following one single specified path.

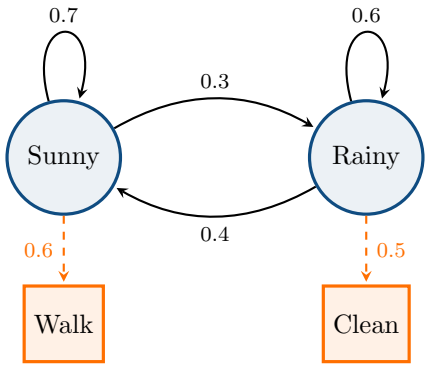
**Single-Path vs Marginal Probability:**

1. If every intermediate state in the sequence is specified, simply **multiply** the transition probabilities along that one path (Examples 4.10, 4.11)
2. If an intermediate state is **not** specified, **sum** the products over every possible value it could take (Example 4.12)
3. This is the same "multiply along a branch, add across branches" rule used in probability trees

## Hidden Markov Models

**Hidden Markov Model (HMM)**

A Hidden Markov Model extends a Markov chain to situations where the true states are **not directly observable**. Instead, at each time step the hidden state probabilistically produces a visible **observation**. An HMM is defined by three components: the **transition probabilities** $A$ between hidden states, the **emission probabilities** $B$ of each hidden state producing each observation, and the **initial probabilities** $\pi$ of starting in each hidden state.




Solid arrows: hidden state transitions ($A$). Dashed arrows: emission probabilities ($B$) to observations.

**Computing the Joint Probability of a Hidden State Path and an Observation Sequence**

**Input:** initial probabilities $\pi$, transition probabilities $A$, emission probabilities $B$, a specific hidden state sequence, an observation sequence of the same length

**Output:** the joint probability $P(\text{states, observations})$

1. Start with $\pi(\text{first state})$.
2. Multiply by the emission probability of the first observation given the first state.
3. For every following time step, multiply by the transition probability from the previous state to the current state, then multiply by the emission probability of the current observation given the current state.
4. Return the final product.

**Example 4.13 — Joint Probability for a Two-Step Path**

An HMM has $\pi(\text{Sunny}) = 0.6$, $\pi(\text{Rainy}) = 0.4$. Transitions: Sunny→Sunny 0.7, Sunny→Rainy 0.3, Rainy→Rainy 0.6, Rainy→Sunny 0.4. Emissions: $P(\text{Walk} \mid \text{Sunny}) = 0.6$, $P(\text{Shop} \mid \text{Sunny}) = 0.3$, $P(\text{Walk} \mid \text{Rainy}) = 0.1$, $P(\text{Shop} \mid \text{Rainy}) = 0.4$. Compute $P(\text{states} = \text{Sunny}, \text{Sunny};\ \text{obs} = \text{Walk}, \text{Shop})$.

**Solution:**

$$\begin{aligned} P &= \pi(\text{Sunny}) \cdot P(\text{Walk} \mid \text{Sunny}) \cdot P(\text{Sunny} \to \text{Sunny}) \cdot P(\text{Shop} \mid \text{Sunny}) \\ &= (0.6)(0.6) \times (0.7)(0.3) \\ &= 0.36 \times 0.21 \end{aligned}$$

$$\boxed{P = 0.0756}$$

**Example 4.14 — Joint Probability for a Different Path**

Using the same HMM as Example 4.13, compute $P(\text{states} = \text{Sunny}, \text{Rainy};\ \text{obs} = \text{Walk}, \text{Clean})$ where $P(\text{Clean} \mid \text{Rainy}) = 0.5$.

**Solution:**

$$\begin{aligned} P &= \pi(\text{Sunny}) \cdot P(\text{Walk} \mid \text{Sunny}) \cdot P(\text{Sunny} \to \text{Rainy}) \cdot P(\text{Clean} \mid \text{Rainy}) \\ &= (0.6)(0.6) \times (0.3)(0.5) \\ &= 0.36 \times 0.15 \end{aligned}$$

$$\boxed{P = 0.054}$$

**Example 4.15 — Joint Probability for a Three-Step Path**

Using the same HMM as Examples 4.13 and 4.14, compute
$P(\text{states} = \text{Rainy}, \text{Rainy}, \text{Sunny};\ \text{obs} = \text{Shop}, \text{Clean}, \text{Walk})$.

**Solution:**

$$\begin{aligned} P &= \pi(\text{Rainy}) \cdot P(\text{Shop} \mid \text{Rainy}) \cdot P(\text{Rainy} \to \text{Rainy}) \cdot P(\text{Clean} \mid \text{Rainy}) \cdot P(\text{Rainy} \to \text{Sunny}) \\ &\cdot P(\text{Walk} \mid \text{Sunny}) \\ &= (0.4)(0.4) \times (0.6)(0.5) \times (0.4)(0.6) \\ &= 0.16 \times 0.3 \times 0.24 \end{aligned}$$

$$\boxed{P = 0.01152}$$

**Markov Chain vs Hidden Markov Model:**

| Aspect | Markov Chain | Hidden Markov Model |
|---|---|---|
| States | Directly observed | Hidden (not observed) |
| What we see | The state sequence itself | Only an observation sequence |
| Parameters needed | Transition probabilities | Transition *and* emission probabilities |
| Typical question | Probability of a state path | Probability of an observation sequence |

# Markov Decision Processes

## Defining a Markov Decision Process

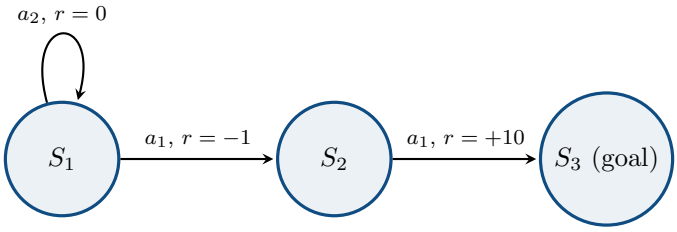
**Markov Decision Process (MDP)**

A Markov Decision Process formally models an RL environment using five components $(S, A, P, R, \gamma)$: a set of **states** $S$, a set of **actions** $A$, **transition probabilities** $P(s' \mid s, a)$ of reaching state $s'$ after taking action $a$ in state $s$, a **reward function** $R(s, a, s')$, and a **discount factor** $\gamma$. The Markov property (Section 4.2.2) applies here too: the next state depends only on the current state and action, not on the full history.

This combines the Markov chain of Section 4.2.2 (states and transitions) with the actions and rewards of reinforcement learning (Section 4.2.1) into one formal framework.

**Policy**

A policy $\pi$ is a rule that specifies which action the agent takes in each state, written $\pi(s) = a$ for a deterministic policy. Solving an MDP means finding the policy that maximizes the expected discounted return from every state.




## Value Functions and the Bellman Equation

**State-Value Function**

The value $V^\pi(s)$ of a state $s$ under policy $\pi$ is the expected discounted return obtained by starting in $s$ and following $\pi$ thereafter. For a deterministic environment (each action leads to exactly one next state), it satisfies the **Bellman equation**:

$$V^\pi(s) = R(s, \pi(s), s') + \gamma\, V^\pi(s')$$

where $s'$ is the state reached from $s$ by taking action $\pi(s)$.

This is simply the discounted-return idea from Section 4.2.1 written *recursively*: the value of a state is this step's reward plus the (discounted) value of wherever you land next.

**Properties of Value Functions**

1. A **goal / terminal state** has value $V(s) = 0$, since no further reward is collected after reaching it *(base case)*
2. Values can be computed **backward** from the goal state toward earlier states *(backward induction)*
3. The **optimal** policy $\pi^*$ at each state chooses the action leading to the highest $R(s, a, s') + \gamma V(s')$ *(greedy w.r.t. the value function)*

**Computing State Values by Backward Induction (Deterministic MDP)**

**Input:** states, one deterministic transition and reward per action, discount factor $\gamma$, a known terminal state

**Output:** $V(s)$ for every state

1. Set $V(\text{terminal state}) = 0$.
2. Identify a state $s$ whose next state's value is already known.
3. Compute $V(s) = R(s, a, s') + \gamma V(s')$ for the action $a$ that will be taken from $s$.
4. Repeat steps 2 to 3, working backward, until every state has a computed value.
5. Return all computed values.

**Example 4.22 — Backward Value Computation on a Three-State Chain**

Using the diagram above ($S_1 \xrightarrow{a_1, r=-1} S_2 \xrightarrow{a_1, r=+10} S_3$, with $S_3$ terminal), compute $V(S_3)$, $V(S_2)$, and $V(S_1)$ using $\gamma = 0.9$.

**Solution:**

$$V(S_3) = 0 \quad \text{(terminal state)}$$
$$V(S_2) = R(S_2, a_1, S_3) + \gamma V(S_3) = 10 + (0.9)(0) = 10$$
$$V(S_1) = R(S_1, a_1, S_2) + \gamma V(S_2) = -1 + (0.9)(10) = -1 + 9$$

$\boxed{V(S_3) = 0, \quad V(S_2) = 10, \quad V(S_1) = 8}$

**Example 4.23 — Choosing Between Two Actions Using Values**

In state $S_1$ from Example 4.22, action $a_1$ leads to $S_2$ (reward $-1$) and action $a_2$ leads back to $S_1$ itself (reward 0, i.e. staying put, so its "next value" is $V(S_1)$ again). Using $V(S_2) = 10$ from Example 4.22 and $\gamma = 0.9$, which action should the optimal policy choose at $S_1$: $a_1$ or $a_2$?

**Solution:**

Value of taking $a_1 = R(S_1, a_1, S_2) + \gamma V(S_2) = -1 + (0.9)(10) = 8$
Value of taking $a_2 = R(S_1, a_2, S_1) + \gamma V(S_1) = 0 + (0.9)(0) = 0$ (using an initial guess $V(S_1) = 0$)

Since $8 > 0$: $\boxed{\text{Optimal action at } S_1 \text{ is } a_1 \text{ (move toward the goal, not stay in place)}}$ This matches the general rule from the theorembox: the optimal policy always picks the action with the higher $R(s, a, s') + \gamma V(s')$. ✓

**Example 4.24 — Backward Value Computation on a Four-State Chain**

A chain has $S_1 \xrightarrow{r=2} S_2 \xrightarrow{r=3} S_3 \xrightarrow{r=5} S_4$(terminal). Compute $V(S_4)$, $V(S_3)$, $V(S_2)$, $V(S_1)$ using $\gamma = 0.5$.

**Solution:**

$$V(S_4) = 0$$
$$V(S_3) = 5 + (0.5)(0) = 5$$
$$V(S_2) = 3 + (0.5)(5) = 3 + 2.5 = 5.5$$
$$V(S_1) = 2 + (0.5)(5.5) = 2 + 2.75$$

$$\boxed{V(S_4) = 0,\ V(S_3) = 5,\ V(S_2) = 5.5,\ V(S_1) = 4.75}$$

**MDP Summary — How Everything in This Chapter Connects:**

1. The **discounted return** $G_t$ (Section 4.2.1) is what a value function $V(s)$ tries to predict *in expectation* from state $s$ onward
2. The **Bellman equation** rewrites that return recursively, one step at a time, instead of summing the whole future at once
3. **Monte Carlo** (Section 4.3) estimates $V(s)$ by averaging actual sampled returns; backward induction here computes $V(s)$ exactly when the model (transitions and rewards) is fully known
4. The **exploration-exploitation** trade-off (Section 4.4) arises precisely because an agent usually does **not** know $P$, $R$, or the true value function in advance, and must learn them through interaction

## Practice Problems

1. A dataset has 40 labeled examples and 160 unlabeled examples. State whether this is a supervised, unsupervised, or semi-supervised learning problem, and justify your answer in one line.

2. A labeled set $L$ for predicting Umbrella use contains: Cloudy→Yes, Cloudy→Yes, Clear→No, Clear→No, Cloudy→No. Compute the initial Naive Bayes parameters $P(\text{Yes})$, $P(\text{No})$, $P(\text{Cloudy} \mid \text{Yes})$, $P(\text{Cloudy} \mid \text{No})$.

3. Using the parameters from the previous question, perform the E-step to compute the soft label $P(\text{Yes} \mid U_1)$ and $P(\text{No} \mid U_1)$ for an unlabeled example $U_1$ with Weather = Cloudy.

4. Using the soft label found in the previous question, perform the M-step to recompute $P(\text{Yes})$ and $P(\text{No})$ (treat $U_1$'s soft label as a fractional addition to the 5 labeled examples).

5. An agent receives rewards $5, 4, 3, 2$ over four consecutive steps. Compute the discounted return $G_0$ using $\gamma = 0.8$.

6. Using the same rewards as the previous question, compute $G_0$ using $\gamma = 1$ (no discounting), and explain in one line why this case is only valid for episodes that are guaranteed to end.

7. A Markov chain has states Hot and Cold with transitions: Hot→Hot 0.6, Hot→Cold 0.4, Cold→Cold 0.7, Cold→Hot 0.3. Given day 1 is Hot, find the probability of the sequence Hot, Cold, Cold.

8. Using the same Markov chain as the previous question, find the probability that day 3 is Hot given that day 1 is Hot (day 2 unknown).

9. An HMM has $\pi(\text{Hot}) = 0.5$, $\pi(\text{Cold}) = 0.5$; transitions as in Question 7; emissions $P(\text{IceCream} \mid \text{Hot}) = 0.7$, $P(\text{Soup} \mid \text{Hot}) = 0.3$, $P(\text{IceCream} \mid \text{Cold}) = 0.2$, $P(\text{Soup} \mid \text{Cold}) = 0.8$. Compute $P(\text{states} = \text{Hot}, \text{Cold};\ \text{obs} = \text{IceCream}, \text{Soup})$.

10. From a fixed starting state $s$, three episodes give total returns $G^{(1)} = 10$, $G^{(2)} = 8$, $G^{(3)} = 6$. Estimate $V(s)$ using Monte Carlo.

11. Two episodes are run from state $s$ with $\gamma = 0.5$. Episode 1 rewards: $6, 4$. Episode 2 rewards: $2, 8$. Estimate $V(s)$ using Monte Carlo.

12. An agent uses $\epsilon = 0.15$. State, for each of the following random draws, whether the agent explores or exploits: (a) $p = 0.10$ (b) $p = 0.50$.

13. An agent has estimated values $Q(\text{Up}) = 4$, $Q(\text{Down}) = 9$, $Q(\text{Left}) = 6$, $Q(\text{Right}) = 2$, with $\epsilon = 0.25$. A random draw gives $p = 0.9$. Which action is chosen, and why?

14. A chain has $S_1 \xrightarrow{r=1} S_2 \xrightarrow{r=4} S_3(\text{terminal})$. Compute $V(S_3)$, $V(S_2)$, and $V(S_1)$ using $\gamma = 0.9$.

15. In a state $S$, action $a_1$ leads to a state with value 12 and reward $-2$, while action $a_2$ leads to a state with value 5 and reward $+3$. Using $\gamma = 0.8$, compute the value of taking each action and determine which action the optimal policy should choose.

# Chapter 5

# Ensemble Learning

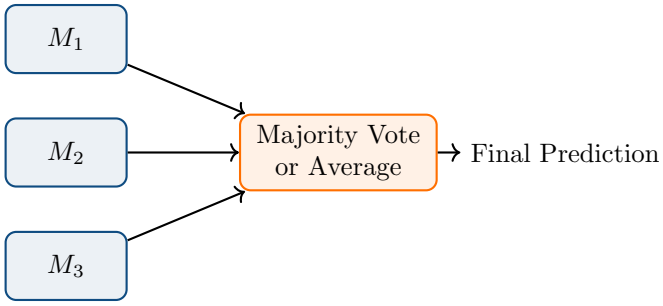
## Introduction to Ensemble Learning

### The Ensemble Idea

**Ensemble Learning**

Ensemble learning combines the predictions of several individual models called **base learners** into one combined prediction. Instead of relying on a single classifier, a **committee** of base learners is trained, and their outputs are combined by **majority vote** for classification or by **averaging** for regression, to produce a final prediction that is usually more accurate than any single base learner alone.

This is the same idea as asking several people for their opinion and going with the majority answer, rather than trusting only one person's judgment.




**Example 5.1 — Majority Vote with Three Classifiers**

Three trained classifiers predict the class of a new patient as: $M_1$ = Sick, $M_2$ = Sick, $M_3$ = Healthy. Combine these using majority vote.

**Solution:** Count the votes for each class.

$$\text{Votes for Sick} = 2$$
$$\text{Votes for Healthy} = 1$$

Sick has more votes. $\boxed{\text{Final prediction: Sick}}$

**Example 5.2 — Majority Vote with Five Classifiers**

Five trained classifiers predict the class of a new email as: Spam, NotSpam, Spam, Spam, NotSpam. Combine these using majority vote.

**Solution:**

$$\text{Votes for Spam} = 3$$
$$\text{Votes for NotSpam} = 2$$

Spam has more votes. Final prediction: Spam An **odd** number of classifiers is often chosen for a two-class problem, for the same reason an odd $k$ is preferred in k-NN (Section 3.4): it prevents an exact tie in the vote.

**Example 5.3 — Averaging Predictions for Regression**

Three regression models predict the price of a house (in lakhs) as: $M_1 = 50$, $M_2 = 54$, $M_3 = 52$. Combine these using averaging.

**Solution:**

$$\text{Combined prediction} = \frac{M_1 + M_2 + M_3}{3}$$
$$= \frac{50 + 54 + 52}{3} = \frac{156}{3}$$

Final prediction = 52 lakhs

**Combining Rules:**

1. **Classification:** combine by **majority vote** (the class chosen by the most base learners), as in Examples 5.1 and 5.2
2. **Regression:** combine by **averaging** the numeric predictions, as in Example 5.3
3. Every base learner contributes equally in a simple vote or average; some advanced ensembles instead use a **weighted** vote, giving more trusted learners more influence

## Why Ensembles Work

**Conditions for an Ensemble to Improve on a Single Classifier**

1. Each base learner must be **better than random guessing** on its own *(accuracy $p > 0.5$ for a two-class problem)*
2. The base learners' mistakes should be as **independent (diverse)** as possible *(errors should not all happen on the same examples)*
3. If both conditions hold, combining by majority vote gives **higher** accuracy than any single base learner

This connects directly to probability: if each of 3 independent classifiers is correct with probability $p$, the ensemble is correct whenever a **majority** (2 or more) is correct, which is a binomial probability calculation.

**Example 5.4 — Ensemble Accuracy When Base Learners Beat Random Guessing**

Three independent classifiers each have accuracy $p = 0.7$. Assuming their errors are independent, compute the accuracy of the majority-vote ensemble.

**Solution:** The ensemble is correct if all 3 are correct, or exactly 2 out of 3 are correct.

$$P(\text{all 3 correct}) = p^3 = (0.7)^3 = 0.343$$

$$P(\text{exactly 2 correct}) = \binom{3}{2} p^2(1-p) = 3(0.49)(0.3) = 0.441$$

$$P(\text{ensemble correct}) = 0.343 + 0.441$$

$\boxed{P(\text{ensemble correct}) = 0.784}$ The ensemble's accuracy (78.4%) is higher than any single classifier's accuracy (70%).

**Example 5.5 — Smaller Improvement**

Three independent classifiers each have accuracy $p = 0.6$. Compute the accuracy of the majority-vote ensemble.

**Solution:**

$$P(\text{all 3 correct}) = (0.6)^3 = 0.216$$

$$P(\text{exactly 2 correct}) = \binom{3}{2} (0.6)^2(0.4) = 3(0.36)(0.4) = 0.432$$

$$P(\text{ensemble correct}) = 0.216 + 0.432$$

$\boxed{P(\text{ensemble correct}) = 0.648}$ The improvement is smaller here (60% → 64.8%) than in Example 5.4 (70% → 78.4%), showing that stronger base learners give the ensemble more to work with.

**Example 5.6 — Ensembling Classifiers Worse than Random Guessing**

Three independent classifiers each have accuracy $p = 0.4$ (worse than random guessing). Compute the accuracy of the majority-vote ensemble.

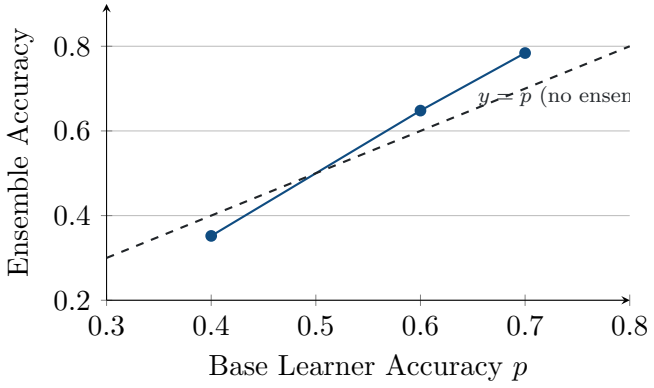
**Solution:**

$$P(\text{all 3 correct}) = (0.4)^3 = 0.064$$

$$P(\text{exactly 2 correct}) = \binom{3}{2} (0.4)^2(0.6) = 3(0.16)(0.6) = 0.288$$

$$P(\text{ensemble correct}) = 0.064 + 0.288$$

$\boxed{P(\text{ensemble correct}) = 0.352}$ Here the ensemble accuracy (35.2%) is even **worse** than a single classifier's accuracy (40%). This confirms Property 1 of the theorembox above: if the base learners are not better than random guessing, combining them does not help and can make things worse. ✓



**The Two Ingredients an Ensemble Needs:**

1. **Accuracy:** each base learner must individually beat random guessing (Examples 5.4 to 5.6 show the ensemble amplifies whatever the base learners already have — good or bad)
2. **Diversity:** the base learners should make **different** mistakes; if every base learner is wrong on exactly the same examples, majority vote gives no benefit at all, since diversity is what allows correct learners to outvote a mistaken one
3. The next two sections cover two different strategies for building a diverse committee of base learners: **Bagging** (Section 5.2) trains learners in parallel on different random samples of the data, and **Boosting** (Section 5.3) trains learners in sequence, each one focusing on the previous learner's mistakes

# Bagging

## Bootstrap Sampling

**Bootstrap Sample**

A bootstrap sample is formed by drawing $n$ examples **with replacement** from an original training set of size $n$. Because the same example can be picked more than once, a bootstrap sample usually contains repeated examples and leaves some original examples out entirely, even though its total size still equals $n$.

This is the same "draw, note it down, put it back, draw again" idea as drawing a card from a deck, recording it, and returning it before the next draw.

**Example 5.7 — Drawing a Bootstrap Sample**

An original training set has 5 examples: $X_1, X_2, X_3, X_4, X_5$. A bootstrap sample is drawn and turns out to be $X_2, X_2, X_4, X_1, X_4$. Identify which original examples are repeated and which are left out.

**Solution:** Count how many times each original example appears in the bootstrap sample.

| Example | Times Selected |
|---|---|
| $X_1$ | 1 |
| $X_2$ | 2 |
| $X_3$ | 0 |
| $X_4$ | 2 |
| $X_5$ | 0 |

Repeated: $X_2, X_4$ Left out: $X_3, X_5$

**Example 5.8 — Probability a Point Is Left Out (n**

An original training set has $n = 4$ examples. One example, $X_1$, is drawn from this set 4 times, with replacement, to build one bootstrap sample. Compute the probability that $X_1$ is never selected.

**Solution:** Each single draw avoids $X_1$ with probability $\frac{n-1}{n} = \frac{3}{4}$, and the 4 draws are independent.

$$P(X_1 \text{ never selected}) = \left(\frac{3}{4}\right)^4 = \frac{81}{256}$$

$P \approx 0.316$

**Example 5.9 — Probability a Point Is Left Out (n**

Repeat Example 5.8 for an original training set of size $n = 5$, drawn 5 times with replacement.

**Solution:**

$$P(X_1 \text{ never selected}) = \left(\frac{4}{5}\right)^5 = \frac{1024}{3125}$$

$\boxed{P \approx 0.328}$ Comparing Examples 5.8 and 5.9, the probability is rising slowly toward a fixed value as $n$ grows. In fact $\left(1 - \frac{1}{n}\right)^n \to e^{-1} \approx 0.368$ as $n$ becomes large.

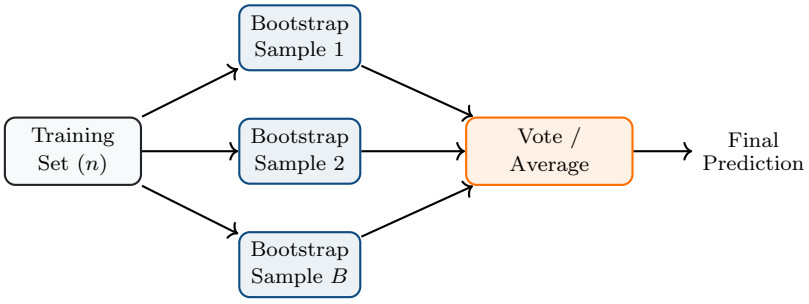
**The 63.2 / 36.8 Rule:**

1. As $n$ grows large, about 63.2% of the original examples appear at least once in a bootstrap sample
2. The remaining 36.8% are left out of that particular bootstrap sample entirely
3. These left-out examples are called **out-of-bag (OOB)** examples for that sample, and are used again in Section 5.2.2

### The Bagging Algorithm

**Bagging**

Bagging (short for Bootstrap Aggregating) trains $B$ base learners independently, each on its own separate bootstrap sample drawn from the same original training set. The $B$ trained models are then combined exactly as in Section 5.1: by **majority vote** for classification, or by **averaging** for regression.




**Bagging Algorithm**

**Input:** training set of $n$ examples, number of bootstrap samples $B$, a base learning algorithm

**Output:** $B$ trained models and a combined prediction rule

1. For $b = 1$ to $B$: draw a bootstrap sample $S_b$ of size $n$, with replacement, from the training set.
2. Train a base learner $M_b$ using only $S_b$.
3. Repeat steps 1 to 2 until all $B$ models are trained.
4. For a new test point, obtain a prediction from every model $M_1, \ldots, M_B$.
5. Combine the $B$ predictions by majority vote (classification) or by averaging (regression).
6. Return the combined prediction.

**Example 5.10 — Combining Bagged Classifiers**

Five decision trees, each trained on a different bootstrap sample, predict the class of a new patient as: Sick, Sick, Healthy, Sick, Healthy. Give the bagging prediction.

**Solution:**

$$\text{Votes for Sick} = 3$$
$$\text{Votes for Healthy} = 2$$

Sick has the majority. $\boxed{\text{Bagging prediction: Sick}}$

**Example 5.11 — Combining Bagged Regression Trees**

Four regression trees, each trained on a different bootstrap sample, predict a house price (lakhs) as: 48, 52, 50, 54. Give the bagging prediction.

**Solution:**

$$\text{Combined prediction} = \frac{48 + 52 + 50 + 54}{4}$$
$$= \frac{204}{4}$$

$\boxed{52 \text{ lakhs}}$

**Example 5.12 — Identifying Out-of-Bag Points**

An original training set has 6 examples $X_1, \ldots, X_6$. One bootstrap sample drawn for model $M_1$ is: $X_3, X_1, X_3, X_6, X_1, X_3$. List the out-of-bag examples for $M_1$.

**Solution:** Selected examples: $X_1$ (yes), $X_2$ (no), $X_3$ (yes), $X_4$ (no), $X_5$ (no), $X_6$ (yes). $\boxed{\text{OOB for } M_1 : X_2, X_4, X_5}$ These 3 out-of-bag points can be used to test $M_1$'s accuracy without needing a separate validation set, since $M_1$ never saw them during training.

**Why Bagging Helps:**

1. Each bootstrap sample is slightly different, so the $B$ trained models make **different** mistakes — the diversity condition from Section 5.1.2
2. Bagging works best with **unstable, high-variance** base learners such as deep, unpruned decision trees, since small changes in the training set noticeably change such a tree
3. **Random Forest** is bagging applied to decision trees, with one extra twist: at each split, only a random subset of attributes is even considered, which further increases diversity between the trees

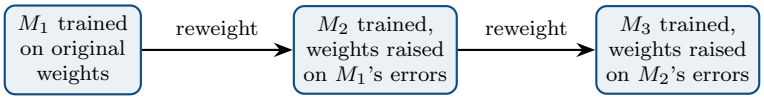
# Boosting

## The Boosting Idea

**Boosting**

Boosting trains base learners **sequentially**, one after another, rather than independently in parallel like bagging. Each new base learner is trained to pay **more attention** to the examples that previous learners got wrong, by increasing the weight of misclassified examples. The final prediction is a **weighted** vote of all the base learners, where more accurate learners get a bigger say.

Unlike bagging, where every bootstrap sample is drawn independently and every model counts equally, boosting builds each model specifically to fix the previous model's mistakes, and trusts more accurate models more when voting.




## Example Weights and Model Weight (AdaBoost-Style)

**Example Weight Update**

Every training example starts with equal weight $w_i = \frac{1}{n}$. After each base learner $M_t$ is trained, examples it **misclassified** have their weight **increased**, and examples it classified **correctly** have their weight **decreased** (or left relatively smaller after renormalizing), so the next learner is forced to focus more on the hard, previously misclassified examples.

**Model (Voting) Weight**

Each trained base learner $M_t$ is given a voting weight $\alpha_t$ based on its (weighted) error rate $\epsilon_t$ on the data it was trained on:

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

A lower error rate $\epsilon_t$ produces a **larger** $\alpha_t$, meaning a more accurate base learner gets more say in the final weighted vote.

This mirrors the ensemble-accuracy calculations of Section 5.1.2: a base learner that is far better than random guessing ($\epsilon_t$ close to 0) deserves much more influence than one that is barely better than a coin flip ($\epsilon_t$ close to 0.5).

**Boosting Algorithm (AdaBoost-Style)**

**Input:** training set of $n$ examples, number of rounds $T$, a base learning algorithm
**Output:** $T$ trained models $M_1, \ldots, M_T$ with voting weights $\alpha_1, \ldots, \alpha_T$

1. Initialize every example's weight to $w_i = \frac{1}{n}$.
2. For $t = 1$ to $T$: train base learner $M_t$ on the training set using the current example weights.
3. Compute $M_t$'s weighted error rate $\epsilon_t$ (the total weight of the examples it misclassified).

4. Compute the model's voting weight $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
5. Increase the weight of every example $M_t$ misclassified, and decrease the weight of every example it classified correctly; then renormalize all weights to sum to 1.
6. Repeat steps 2 to 5 for all $T$ rounds.
7. For a new test point, combine all $T$ predictions using a weighted vote: $\sum_t \alpha_t \cdot$ [prediction of $M_t$].
8. Return the combined prediction.

**Example 5.13 — Computing Voting Weight from Error Rate**

A base learner $M_1$ has weighted error rate $\epsilon_1 = 0.2$. Compute its voting weight $\alpha_1$. (Use $\ln 4 \approx 1.386$.)

**Solution:**

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-0.2}{0.2}\right) = \frac{1}{2}\ln\left(\frac{0.8}{0.2}\right) = \frac{1}{2}\ln(4)$$

$\boxed{\alpha_1 \approx 0.693}$

**Example 5.14 — Voting Weight for a Weaker Learner**

A base learner $M_2$ has weighted error rate $\epsilon_2 = 0.4$. Compute its voting weight $\alpha_2$. (Use $\ln 1.5 \approx 0.405$.)

**Solution:**

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-0.4}{0.4}\right) = \frac{1}{2}\ln\left(\frac{0.6}{0.4}\right) = \frac{1}{2}\ln(1.5)$$

$\boxed{\alpha_2 \approx 0.203}$ Comparing to Example 5.13: $M_1$ (error 0.2) receives a much higher voting weight $\alpha_1 \approx 0.693$ than $M_2$ (error 0.4) with $\alpha_2 \approx 0.203$, exactly as expected from the definition — the more accurate learner earns more influence.

**Example 5.15 — Combining Weighted Votes**

Three trained learners predict the class of a new example as: $M_1 = +1$ (with $\alpha_1 = 0.8$), $M_2 = -1$ (with $\alpha_2 = 0.5$), $M_3 = +1$ (with $\alpha_3 = 0.3$). Compute the final weighted-vote prediction.

**Solution:** Sum the voting weights separately for each predicted class.

$$\text{Weight for } +1 = \alpha_1 + \alpha_3 = 0.8 + 0.3 = 1.1$$
$$\text{Weight for } -1 = \alpha_2 = 0.5$$

Since $1.1 > 0.5$: $\boxed{\text{Final prediction: } +1}$

**Bagging vs Boosting:**

| Aspect | Bagging | Boosting |
|---|---|---|
| Training order | Parallel, independent | Sequential, each depends on the last |
| Sampling | Bootstrap (random, with replacement) | Full data, but example weights change each round |
| Combining | Simple majority vote / average | Weighted vote using each $\alpha_t$ |
| Main goal | Reduce variance | Reduce bias (and variance) |
| Best base learner | High-variance (deep trees) | Weak learner (shallow trees, "stumps") |
| Risk | Less prone to overfitting | Can overfit if run for too many rounds |

**Example 5.16 — Tracing Example Weight Changes Across One Round**

Four training examples start with equal weight $w_i = 0.25$. Base learner $M_1$ misclassifies $X_2$ and $X_4$, and correctly classifies $X_1$ and $X_3$. If misclassified examples have their weight doubled and correctly classified examples have their weight halved, find the new weights before renormalization, and then the renormalized weights.

**Solution:**

$$w_1' = 0.25 \times 0.5 = 0.125 \text{ (correct)}$$
$$w_2' = 0.25 \times 2 = 0.5 \text{ (misclassified)}$$
$$w_3' = 0.25 \times 0.5 = 0.125 \text{ (correct)}$$
$$w_4' = 0.25 \times 2 = 0.5 \text{ (misclassified)}$$
$$\text{Sum} = 0.125 + 0.5 + 0.125 + 0.5 = 1.25$$

Renormalize by dividing each weight by the sum 1.25:

$$w_1'' = \frac{0.125}{1.25} = 0.1, \quad w_2'' = \frac{0.5}{1.25} = 0.4, \quad w_3'' = \frac{0.125}{1.25} = 0.1, \quad w_4'' = \frac{0.5}{1.25} = 0.4$$

$\boxed{\text{New weights: } w_1 = 0.1,\ w_2 = 0.4,\ w_3 = 0.1,\ w_4 = 0.4}$ $X_2$ and $X_4$ (misclassified) now carry far more weight (0.4 each) than $X_1$ and $X_3$ (0.1 each), so the next base learner $M_2$ will be trained to focus much more heavily on getting $X_2$ and $X_4$ correct.

## Practice Problems

1. Three trained classifiers predict a new example's class as: Yes, No, Yes. Combine these predictions using majority vote.

2. Five regression models predict a car's price (lakhs) as: 12, 14, 13, 15, 11. Combine these predictions using averaging.

3. Explain in 2 to 3 lines why an ensemble of classifiers, each with individual accuracy below 50%, can end up performing worse than any single classifier.

4. Three independent classifiers each have accuracy $p = 0.8$. Compute the accuracy of the majority-vote ensemble (correct when all 3, or exactly 2 of 3, are correct).

5. Three independent classifiers each have accuracy $p = 0.5$. Compute the accuracy of the majority-vote ensemble, and explain what this result shows about the accuracy condition needed for ensembles to help.

6. An original training set has 6 examples $Y_1, \ldots, Y_6$. A bootstrap sample drawn is: $Y_2, Y_5, Y_2, Y_2, Y_6, Y_5$. List which examples are repeated and which are left out (out-of-bag).

7. For an original training set of size $n = 6$, drawn 6 times with replacement to build one bootstrap sample, compute the probability that a specific example $Y_1$ is never selected.

8. Four decision trees, each trained on a separate bootstrap sample, predict a new patient's class as: Sick, Healthy, Sick, Sick. Give the bagging prediction.

9. Five regression trees trained via bagging predict a house price (lakhs) as: 60, 58, 62, 61, 59. Give the bagging prediction.

10. Explain in 2 to 3 lines why bagging works best with high-variance base learners such as deep, unpruned decision trees rather than with already-stable models.

11. A base learner has weighted error rate $\epsilon = 0.1$. Compute its voting weight $\alpha$. (Use $\ln 9 \approx 2.197$.)

12. A base learner has weighted error rate $\epsilon = 0.5$. Compute its voting weight $\alpha$, and explain in one line what this result means for how much this learner should be trusted in the final vote.

13. Four boosted learners predict a new example's class with the following (prediction, voting weight) pairs: $(+1, 0.6)$, $(-1, 0.9)$, $(+1, 0.4)$, $(-1, 0.3)$. Compute the final weighted-vote prediction.

14. Three training examples start with equal weight $w_i = \frac{1}{3}$. A base learner misclassifies only $X_2$. If the misclassified example's weight is tripled and the correctly classified examples' weights are left unchanged, find the renormalized weights for all three examples.

15. List three differences between Bagging and Boosting, covering training order, how each new base learner is influenced by earlier ones, and how their final predictions are combined.