# Applied Statistics - Notes - v0.4.0

260236

June 2025

# Preface

Every theory section in these notes has been taken from two sources:

- The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. [2]

- An Introduction to Statistical Learning: with Applications in Python. [3]

- Applied Multivariate Statistical Analysis. [4]

- Course slides. [5]

About:

 GitHub repository

These notes are an unofficial resource and shouldn't replace the course material or any other book on applied statistics. It is not made for commercial purposes. I've made the following notes to help me improve my knowledge and maybe it can be helpful for everyone.

As I have highlighted, a student should choose the teacher's material or a book on the topic. These notes can only be a helpful material.

# Contents

# 1    Business Data Analytics

## 1.1    Multivariate Descriptive Statistics

When we move from **analyzing** a single variable (univariate analysis) to **multiple variables at once**, we enter the realm of **Multivariate (MV) analysis**. A natural question arises: *Is multivariate analysis just a replication of univariate analysis across several variables?*

The answer is no, multivariate analysis introduces new and fundamental questions that cannot be answered by simply analyzing variables individually. The **core focus** shifts to **understanding how these variables interact with each other**. Specifically, we are concerned with the **dependence** and **correlation between variables**.

### ▤ Covariance: Measuring Joint Variability

**To capture how two variables vary together**, we use **Covariance**. The **Sample Covariance** between variables $x_j$ and $x_k$ is calculated as:

$$\text{cov}_{jk} = \text{Cov}(x_j, x_k) = s_{jk} = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \tag{1}$$

- $s_{jk} = 0 \Rightarrow$ implies that there is **no linear relationship** between the two variables.

- $s_{jk} > 0 \Rightarrow$ suggests that **as one variable increases**, the **other tends to increase**.

- $s_{jk} < 0 \Rightarrow$ one **variable** tends to **decrease when the other increases**.

### ⚠ Covariance Is Not Standardized

The **value of covariance is not standardized**, it **depends on the units of measurement**, which makes comparisons difficult. For example:

- Suppose we're measuring

  - Height in centimeters
  - Weight in kilograms

- The covariance between height and weight will be expressed in *centimeter-kilogram* units.

Now imagine we convert height to meters. The covariance value changes, because now we're multiplying meters × kilograms instead of centimeters × kilograms. Even though the **relationship between height and weight hasn't changed**, the **numerical value of covariance does change due to this unit change. Because of unit dependency, it's hard to compare covariances between different variable pairs**. Finally, it is **hard to interpret the magnitude of covariance in any absolute sense** (e.g., is 50 a large covariance or small? It depends on the units!).

### ✅ Correlation: Standardized Covariance

To standardize covariance and **measure the strength of a linear relationship** on a scale between $-1$ and $1$, we use the **Correlation** coefficient, defined as:

$$\text{cor}_{jk} = r_{jk} = \frac{s_{jk}}{\sqrt{s_{jj}}\sqrt{s_{kk}}} = \frac{\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)\left(x_{ik} - \bar{x}_k\right)}{\sqrt{\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2}\sqrt{\sum_{i=1}^{n}\left(x_{ik} - \bar{x}_k\right)^2}} \tag{2}$$

This formula divides the covariance by the product of the standard deviations of the two variables, giving a **unitless value**:

- $r = 0$: No linear correlation

- $r > 0$: Positive correlation (**both variables increase or decrease together**)

- $r < 0$: Negative correlation (**one increases while the other decreases**)

- $|r| = 1$: Perfect correlation (**exact linear dependence**)

Thus, correlation not only **reveals the direction of the relationship** but also its **strength**.



Figure 1: Direction of correlation: positive (left), none (center), negative (right).

### ▤ Describing MV Data: Vector and Matrices

When analyzing multivariate data:

- We compute the **vector of Sample Means**:

$$\bar{\mathbf{X}} = [\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_p] \tag{3}$$

- And the **symmetric and positive semidefinite** (eigenvalues are non-negative) **Variance-Covariance matrix S**, which **summarizes the covariances between all pairs of variables**:

$$S = \begin{bmatrix} s_{11} & \cdots & s_{1p} \\ \vdots & \ddots & \vdots \\ s_{p1} & \cdots & s_{pp} \end{bmatrix} \tag{4}$$

- Alternatively, we can use the **Correlation matrix R**, where **all diagonal elements are 1** (because each variable is perfectly correlated with itself) and **off-diagonal elements are correlation coefficients**:

$$R = \begin{bmatrix} 1 & \cdots & r_{1p} \\ \vdots & \ddots & \vdots \\ r_{p1} & \cdots & 1 \end{bmatrix} \tag{5}$$

### 📗 Scatterplots - Visualizing Variable Pairs

One of the most intuitive and widely used tools in multivariate analysis is the **2D Scatterplot**. Each plot shows how two variables relate to each other:

✔ Clusters or linear trends can indicate correlation or dependence.

✔ Scatterplots are **ideal for spotting positive, negative, or no correlation** visually.

However, scatterplots have a **limitation**: they **only** allow us to **analyze two variables at a time**. When dealing with many variables, the **number of possible pairings becomes large**, making it **difficult to read or interpret** the scatterplots individually. This is where quantitative measures (like correlation matrices) and higher-dimensional graphics come into play.



Figure 2: Scatterplot matrix of four variables. This scatterplot matrix displays all pairwise relationships among four variables:

- Diagonal plots (top-left to bottom-right): Histograms showing the distribution of each variable.

- Off-diagonal plots: 2D scatterplots illustrating the relationship between each pair of variables.

### 🗐 Rotated Plots in 3D - Capturing Complexity

When dealing with **three variables**, we can **extend scatterplots into 3D space** using **Rotated plots**. These visualizations allow us to:

- ✔ **Explore interdependencies** among three variables at once.

- ✔ **Gain spatial insight** into how data points spread in three-dimensional space.

- ✔ Observe **complex patterns that are invisible in 2D**.

Yet again, as we move **beyond three variables**, visualizing becomes **impractical**, our brains cannot easily comprehend 4D or higher dimensions. Hence, dimensionality reduction techniques like PCA are often used alongside visualizations to make high-dimensional data "digestible".



Figure 3: A simple rotated plots in 3D.

### 📗 Star Plots - Shape-Based Comparison

**Star plots** offer a creative way to **represent multivariate data**:

- **Each variable is represented as a ray** (spoke) starting from a central point.

- The **length of each ray** corresponds to the **value** of that variable.

- When **rays are connected**, they form a "star-like shape" **unique** to each observation.

This method is **excellent for comparing patterns** between observations:

- ✔ **Similar shapes** suggest **similar data profiles**.

- ✔ **Differences** in shape can **quickly highlight outliers** or clusters.

However, star plots have limitations:

- ✘ They **do not quantify correlation**.

- ✘ The **direction** and **magnitude** of relationships between variables are **not explicit**.

- ✘ They are better for **visual pattern recognition** than for precise statistical analysis.



Figure 4: Star Plot (Radar Chart) of Multivariate Data.

### 🗐 Chernoff Faces - Human-Centric Visualization

Chernoff faces [1] are an **innovative visualization method** where **multivariate data is represented as a human face**:

- Each **variable controls a facial feature** (e.g., mouth curvature, eye size, nose length).

- **People are naturally attuned to recognizing faces** and subtle differences in expressions.

- Hence, Chernoff faces **leverage human perception** for quickly comparing **multivariate observations**.

Despite being engaging, Chernoff faces also have **drawbacks**:

- ✖ They **do not provide numerical precision**.

- ✖ The **mapping of variables to facial features** can be **arbitrary**.

- ✖ They work best as a **qualitative summary tool** rather than for deep statistical inference.



Figure 5: Some Chernoff faces. [1]

| Graphic Type | Strengths | Limitations |
| --- | --- | --- |
| Scatterplots | Clear view of pairwise relationships | Hard to scale beyond 2 variables |
| 3D Plots | Visualizes 3-variable interaction | Limited to 3 dimensions, requires rotation |
| Star Plots | Quick shape-based comparison across variables | No quantification, poor at showing correlations |
| Chernoff Faces | Leverages facial perception for comparison | Subjective, lacks precision |

Table 1: When and why to use graphics.

## 1.2   Dimensionality Reduction

### ⚠ The Challenge: Data in High Dimensions

In many real-world problems, we **collect multiple variables for each observation**. For example, in a medical study, a patient might be described by age, weight, blood pressure, and dozens of test results. This leads to **high-dimensional data**, where **each observation is a point in a complex, multi-dimensional space** (formally, a Euclidean space of dimension $p$).

*The problem?* As the **number of variables** ($p$) **increases**, the **data becomes harder to visualize, interpret, and model**:

- ❗ Some **variables** might be **redundant** or **highly correlated**.

- ❗ **Computations** become more **expensive**.

- ❗ **Patterns become obscured** by the complexity.

### 🏁 Goal of Dimensionality Reduction

We want to **summarize the data using fewer variables**, say $k$ derived variables (with $k < p$), that still **retain most of the information**. This process is a balancing act:

- **Clarity**: fewer variables make data easier to understand and visualize.

- **Risk of oversimplification**: reducing dimensions too much can cause loss of important information.

The key concept here is that in statistics, **information is variability**. The more variability we retain from the original data, the more information we preserve.

---

**Example 1: Blood Cells**

Imagine we measure thickness and diameter for a set of red blood cells:

- Each cell = one observation with two variables.

- We can represent this as a table (numbers) or as a 2D scatterplot.

Now we ask ourselves: *Can we describe these cells using only one feature instead of two?*
If we choose only diameter or only thickness, we'll lose detail:

- Some cells will appear more similar than they really are.

- We miss variability that distinguishes them.

So, we seek a better single feature, one that captures the most variation possible from both thickness and diameter combined.

---

### 📈 The Statistical Insight: Maximize Variability

Rather than randomly picking a feature, we **analyze the directions along which the data varies the most**.

1. First, we find the **direction of maximum spread**.

2. Then, the **second most spread direction**, orthogonal to the first.

This is the essence of **Principal Component Analysis (PCA)**, a **dimensionality reduction technique that finds the best directions** (linear combinations of variables) **to project the data**, while **maximizing retained variability**.

### 📄 Dimensionality Reduction in Practice

Let's formalize the idea:

- We start with a **data matrix** $X$ of shape $n \times p$ ($n$ observations, $p$ variables).

- The <mark>goal</mark> is to **obtain a new matrix** $M$ of shape $n \times k$, with $k < p$, that **captures most of the variability**.

- The **difference** between $X$ and $M$ is **residual variation**, the **information lost**.

In summary, **Dimensionality Reduction** is about simplifying complexity: **transforming a large set of variables into a smaller**, more interpretable set **without losing the essence of the data**. It's central to data exploration, preprocessing, and modeling, especially when working with high-dimensional datasets.

## 1.3   Principal Component Analysis

### ✅ Why PCA?

Imagine we have a data set with **many variables**, such as measurements of people, products, or cities. Some **variables might be closely related** (like height and weight), **while others might carry similar information**. This creates two challenges:

1. It becomes **hard to analyze and interpret** the data.

2. **Redundancy** can lead to inefficiency and confusion.

**Principal Component Analysis (PCA)** helps solve this by providing a **simplified version of the dataset**, where we focus only on the **essential information**.

### 📘 The Main Idea

PCA works by **creating new variables**, called **principal components**, that are:

- **Combinations** of the original variables.

- **Ordered** so that the **first component captures the most variability** in the data.

- Each **subsequent component captures the next most variability**, but only from what's left over, and each **new component is uncorrelated with the previous** ones.

Imagine this like finding better "angles" or "perspectives" from which to view our data, ones that maximize how much we can see (i.e., variability) with as few perspectives as possible.

### 📘 Variability is Information

In statistics, **Variability**, **how much values change from one observation to the next**, is considered **information**. The more variability a component captures, the **more useful it is** in understanding the data. So, PCA's job is to **find the directions in which the data varies the most**, and to use those directions to summarize the dataset.

### ❓ How PCA Changes the Dataset

Let's say we start with $p$ variables (like height, weight, age, income...). **PCA gives us $p$ principal components**, but the *magic* is that **we usually only need the first few** to understand most of what's going on.

So instead of working with the full, original dataset, we now have a **simpler version**:

- We still have the **same number of observations**.

- But each observation is now described by **fewer, more informative variables** (components).

This is called a **low-dimensional representation** of the data.

### 📊 Scores and Loadings - The Ingredients and the Result

In this new system:

- The **Scores** tell us where **each observation lies along the new axes** (the principal components).

- The **Loadings** tell us **how the original variables contribute to each component**.

Think of it like cooking:

- **Original variables** = ingredients

- **Loadings** = recipe instructions

- **Principal components** = final dishes

- **Scores** = ratings of the dishes for each person (observation)

### √x̄ Matrix Representation

Let's denote the data matrix as $X$, with $n$ rows (observations) and $p$ columns (variables). PCA is essentially about **transforming this matrix $X$ into a new matrix**, where:

- The data is now described by **principal components** instead of the original variables.

- The **goal is to rotate and simplify** the data in a way that **emphasizes the most important directions** (maximum variability).

Two key matrices in PCA:

- **Loadings Matrix ($V$)**

    - This matrix contains the **weights** used to build the principal components.

- Each **column** in V corresponds to a **principal component**.
- Each **value** in $V$ shows **how much each original variable contributes** to the component.

We can think of $V$ as the recipe book: it tells us how to combine original variables to create the new components.

- **Scores Matrix ($U$)**

  - This matrix contains the **projections of the data** onto the principal components.
  - Each **row** in $U$ represents an **observation in the new PCA space**.
  - These are called scores, they tell us **where each observation lands along the new axes** (PC1, PC2...).

So $U$ is the result: it shows **how our data looks in the new, simplified system**.

PCA can be **computed using Singular Value Decomposition (SVD)**:

$$X = U \cdot S \cdot V^T \tag{6}$$

PCA **factorizes the data matrix** $X$. Here's what each matrix means:

- $X$: Original data ($n \times p$)

- $U$: Scores matrix ($n \times p$), orthogonal (columns are independent)

- $S$: Diagonal matrix of **singular values** (related to the variance explained)

- $V^T$: Transposed loadings matrix ($p \times p$), also orthogonal.

But for understanding, focus on this **simpler form**:

$$\textbf{PCA result} = \text{Scores} = X \cdot V$$

We **multiply the data $X$ by the loadings matrix $V$** to obtain the **scores**.

However, both $U$ and $V$ are **orthogonal matrices**, meaning:

- Their columns are perpendicular (**no redundancy**).

- Principal **components are uncorrelated**, each new component captures new, non-overlapping information.

This is mathematically elegant and practically useful because it **removes multicollinearity** and makes downstream **analysis simpler and more robust**.

### ❷ How Much Information Do We Keep?

Each principal component has a **percentage of variance explained**, this tells us how much of the original data's information it retains. Often, **the first 1 or 2 components explain so much that we can ignore the rest**.

In conclusion, **PCA helps us focus on what matters** in our data. It's like cleaning our glasses: everything becomes sharper, simpler, and more meaningful. We go from being overwhelmed by numbers to seeing clear patterns. It's a tool used everywhere, from finance to biology, marketing to engineering, whenever people need to make sense of complex data.

## 1.4  PCA Reference System

### ❓ Why Do We Talk About Projections in PCA?

When we say that PCA projects data, we mean it **transforms the data points by placing them onto new axes**, the principal components, that better represent the structure of variability.

❓ *But why are projections so important?* Because PCA has **two goals**:

1. **Maximize variance**: We want the **data to spread out as much as possible along the new axis**. This spread means we're capturing differences between observations.

2. **Minimize residuals**: We want to **minimize the error** when we approximate each point using the new axes. This is done by projecting each point perpendicularly onto the new axes, just like the shortest path between a point and a line.

This is why we "keep talking about projections": they allow us to **retain the most information** with **the least distortion**.

### ☰ Generalizing to More Dimensions

In real datasets, we often have **more than two variables**. PCA scales to $p$ dimensions, and the idea of projection still applies:

- PC1: The **direction in $p$-dimensional space** where data varies most.

- PC2: The next direction, **perpendicular to the first**, that captures remaining variability.

- This continues until we have $p$ principal components, each orthogonal to the others.

So **PCA rotates the entire dataset into a new reference system** where the data structure is easier to understand.

### 🗐 A New Reference System

After PCA, our data now lives in a **new coordinate system**:

- The **original variables** (e.g., height, weight) are no longer the axes.

- Instead, the axes are **principal components**, which are **combinations of the original variables**.

This new reference system has two main features:

1. **PCA results are rotation invariant**

   🗐 *What does this means?* It means that if we rotate our dataset (e.g., by changing the coordinate system), PCA still finds the same underlying structure, the same principal components (relative to the data).

❓ **Why?** PCA depends only on the relationships between the data points, specifically: the variances and covariances between variables. These are not affected by rotations of the data in space. Mathematically, PCA extracts eigenvectors of the covariance matrix that are invariant under rotation with respect to relative directions.

2. **Principal Components are independent (uncorrelated)**

📄 **What does this means?** The principal components (PC1, PC2, ...) are uncorrelated with each other. Knowing the value of one PC tells us nothing about the value of the next.

❓ **Why?** Each new component is orthogonal to the previous one. And since the correlation is equal to the cosine of the angle between the variables $(\cos(90) = 0)$, PAC ensures that the correlation between the PCs is zero.

✔ In high dimensions, **collinearity** often indicates redundant information between variables and can cause problems in the analysis. **PCA solves** this problem by giving us independent, uncorrelated variables.



Figure 6: We have a bunch of scattered red dots. We have drawn a black line through the middle of the cloud of dots in the direction where the dots are most scattered. This line is our first principal component (PC1). For each red point, we drop a perpendicular line onto the black line. Where it lands, we place a blue point. This is called projection. Furthermore, the longer the black line, the more red points are taken into account (thus maximizing variance); on the other hand, the blue points must be close to the red points (shorter the distance) to minimize residuals and lose less information.

17

Figure 7: By drawing two lines, PC1 and PC2, we obtain a new reference system. It is a rotated system. The axes are principal components, which are combinations of the original variables.

### 🛠 Principal Components: Linear Combinations

Let's now talk about **how principal components are constructed**. Each principal component is a **linear combination of the original variables**. This means:

$$\text{PC}_1 = \phi_{11} \cdot x_1 + \phi_{21} \cdot x_2 + \cdots + \phi_{p1} \cdot x_p$$

But in general:

$$\text{PC}_j = \phi_{1j} \cdot x_1 + \phi_{2j} \cdot x_2 + \cdots + \phi_{pj} \cdot x_p \tag{7}$$

Where:

- The $\phi$-values (phi) are called **Loadings**.

- They are the **weights assigned to each original variable**.

- The **higher the loading**, the **more that variable contributes** to that principal component.

For **each observation** (e.g., a person, product), we now **compute their position in the new PCA space**:

$$z_{ij} = \phi_{1j} \cdot x_{i1} + \phi_{2j} \cdot x_{i2} + \cdots + \phi_{pj} \cdot x_{ip} \tag{8}$$

These values $z_{ij}$ are called **Scores**. They tell us:

- Where each observation lies **along a principal component axis**.

- How much that **component contributes** to describing this observation.

So now, instead of describing each person with $p$ original variables, we describe them with $k$ scores, where $k < p$, but we still capture the essence of their data.

## 1.5   PCA as Optimization Problem

◎ **What Is PCA Trying to Achieve?**

PCA aims to find new variables (**principal components**) that are:

- **Linear combinations** of the original variables.

- Chosen so that the **first principal component** (PC1) captures the **maximum possible variance** in the data.

- **Second and further PCs** capture the **remaining variance**, while being uncorrelated with all previous ones.

In other words, ==PCA tries to find a new axis (direction) along which the data varies the most==. This is the first principal component. Once this direction is found, each data point can be projected onto it to create a simplified representation of the data.

==To find this direction== (**PCA1**), **PCA solves a mathematical optimization problem**. This is because the **goal is to find the direction that maximizes the variance** of the data projected onto that direction (we want to maximize the variance because it captures the most variability, information, in the data).

❷ **How Is This Formulated as an Optimization Problem? (PC1)**

1. **Define a New Variable $Z_1$**. Let's construct a new variable $Z_1$ which is a **linear combination** of the original variables:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + \cdots + \phi_{p1}X_p$$

   Where:

   - $X_j$ is the $j$-th variable (all observations for that variable)
   - $\phi_{j1}$ is the $j$-th weight (loading) assigned to the variable $X_j$.

   The subscript 1 indicates that they belong to the first principal component.

2. **Objective - Maximize Variance of $Z_1$**. We want to maximize the variance to capture the most variability (information) in the data:

$$\text{Maximize} \quad \text{Var}(Z_1) = \text{Var}(\phi_1^T X)$$

   Where:

   - $X$ is the **data matrix** $n \times p$
   - $\phi_1$ is the vector $p \times 1$ of **weights** (**loadings**) used to build $Z_1$. It is transposed to allow multiplication with the data matrix $X$.

   More specifically:

$$\text{Var}(Z_1) = \frac{1}{n}\sum_{i=1}^{n} z_{i1}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}\cdot x_{ij}\right)^2$$

Where $z_{i1}$ is the **score** (see page 18) of the $i$-th observation on the **first principal component** (e.g., $z_{i2}$ score of observation $i$ on PC2). In our case it is:

$$z_{i1} = \sum_{j=1}^{p} \phi_{j1} \cdot x_{ij}$$

3. **Constraint - Normalize the Loadings**. We need a **constraint** because we could scale the weights (loadings) infinitely. Therefore, we need **to fix the length** (norm) **of the loadings vector**.

$$\sum_{j=1}^{p} \phi_{j1}^2 = 1$$

Or, in vector notation:

$$||\phi_1||^2 = 1$$

The constraint says that the total energy or length of the loadings vector is fixed to 1 and PCA can only choose the direction of $\phi_1$, not its size.

Now we can write the **full optimization problem for PC1**:

$$\text{Maximize} \quad \text{Var}(Z_1) \quad \text{subject to} \quad ||\phi_1||^2 = 1 \tag{9}$$

Specifically, the extended form:

$$\max_{\phi_{11},\ldots,\phi_{p1}} \frac{1}{n} \sum_{i=1}^{n} \left( \sum_{j=1}^{p} \phi_{j1} \cdot x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \phi_{j1}^2 = 1 \tag{10}$$

**❷ Okay, but how does PCA solve the optimization problem?**

In previous steps, we posed PCA as:

- Maximize the variance of $Z_1 = \phi_1^T X$

- Subject to $||\phi_1||^2 = 1$

The solution to this problem: the **optimal loadings vector** $\phi_1$ **(for PC1) is the first eigenvector of the sample covariance matrix S**. Let's analyze this sentence to understand how and why.

> **❷ Why Eigenvectors Help Solve This?** We need eigenvectors because PCA's optimization problem is mathematically equivalent to a linear algebra problem whose solution requires eigenvectors.
>
> The **mathematical problem can also be written in quadratic form**[1]
>
> $$\text{Var}\,(Z_1) = \phi_1^T S \phi_1$$
>
> Where $S$ is the covariance matrix (page 6). It follows the exact pattern of the quadratic form, because $\phi_1$ is a vector of loadings, $S$ is the covariance matrix (square), and the whole expression evaluates to a scalar: the variance of $Z_1$.
>
> In linear algebra, the **quadratic form problem is solved by eigenvectors**. Using the **Rayleigh Quotient Theorem**, we can obtain the goal of PCA. Given a symmetric matrix $S$ (like the covariance matrix in PCA), the *maximum value* of the quadratic form:
>
> $$\phi_1^T S \phi_1 \quad \text{subject to} \quad ||\phi_1||^2 = 1$$
>
> Is achieved when $\phi_1$ is the eigenvector corresponding to the largest eigenvalue of $S$. Therefore, the **solution must be the first eigenvector** (by the Rayleigh Quotient Theorem).

---

[1]A **Quadratic Form** is any expression that involves a vector, a matrix, and the same vector transposed. The general structure is:

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{A}\mathbf{x} \tag{11}$$

Where $\mathbf{x}$ is a vector, $\mathbf{A}$ is a square matrix, and the result is a scalar.

**❷ Optimization Problem for the Second Principal Component (PC2)**

The following steps allow to obtain the PC2. We avoid a long explanation here, because we made the same logical steps of the optimization problem for PC1.

1. **Define a New Variable $Z_2$.** Just like with PC1, we define PC2 as a **linear combination** of the original variables:

$$Z_2 = \phi_{12}X_1 + \phi_{22}X_2 + \cdots + \phi_{p2}X_p$$

For a specific observation $i$:

$$z_{i2} = \sum_{j=1}^{p} \phi_{j2} \cdot x_{ij}$$

Where: $\phi_2 = [\phi_{12}, \ldots, \phi_{p2}]^T$ is loadings vector for PC2. Here the **goal is to find the best $\phi_2$ to create PC2**.

2. **Objective - Maximize Variance of $Z_2$.** Just like PC1, we want to **maximize variance**:

$$\max_{\phi_2} \quad \mathrm{Var}\,(Z_2) = \frac{1}{n}\sum_{i=1}^{n} z_{i2}^2$$

3. **Constraints (there are two now)**

   (a) **First Constraint: Normalize the Loadings**. We must prevent arbitrary scaling of the weights:

   $$\sum_{j=1}^{p} \phi_{j2}^2 = 1 \quad \text{or} \quad \|\phi_2\|^2 = 1$$

   (b) **Second Constraint: Ensure PC2 is Uncorrelated with PC1**. This is new compared to PC1. Here we want the covariance between PC1 and PC2 to be zero; in other words, we're saying **we don't want these two results to be correlated**.

   $$\mathrm{Cov}(Z_1, Z_2) = 0$$

   Mathematically, this is equivalent to saying:

   $$\phi_1^T \phi_2 = 0 \qquad \text{(orthogonality)}$$

   If **two directions are orthogonal**, their **projections** (scores) are **uncorrelated**.

Finally, we can write the **full optimization problem for PC2**:

$$\max_{\phi_{12}, \ldots, \phi_{p2}} \quad \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j2}x_{ij}\right)^2 \tag{12}$$

**Subject to**:

$$\sum_{j=1}^{p} \phi_{j2}^2 = 1 \quad \text{and} \quad \sum_{j=1}^{p} \phi_{j1} \cdot \phi_{j2} = 0 \tag{13}$$

❓ **And how to solve the PC2 optimization problem?** The solution to this constrained optimization problem is similar to the PC1:

$$\phi_2 = \mathbf{e}_2 \tag{14}$$

Where:

- $\mathbf{e}_2$ equal to the **second eigenvector** of the **covariance matrix** $S$.

- This eigenvector corresponds to the **second largest eigenvalue** $\lambda_2$.

## 1.6    Proportion of Variance Explained (PVE)

In the previous sections, we explained how to compute PC1, PC2, etc. as new variables. Each PC is a linear combination of the original variables, and each maximizes the variance under orthogonality constraints. But now the question becomes: "*okay, we computed these new components... but how useful are they?*".

In PCA, variance equals information. So now our **goal** is: "how much **information (variance) is explained by PC1**? And by **PC2**? And by **PC3**? And so on". This is where **Proportion of Variance Explained (PVE)** comes in.

We know that each principal component explains some variance:

- PC1 captures the **largest possible amount** of variance.

- PC2 captures the **next largest amount**, and so on.

- All PCs **together** explain **100% of the variance** in the data (no information is lost if we keep all of them).

If the data matrix $X$ is centered (i.e., variables have mean 0), then the **total variance in the data** is:

$$\text{Var}_{\text{TOT}} = \sum_{j=1}^{p} \text{Var}(X_j) = \sum_{j=1}^{p} \left( \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 \right) \tag{15}$$

This is just the **sum of the variances** of the original variables.

To know the **variance** at the step $k$, so for the $k$**-th principal component**, we generalize:

$$\text{Var}_k = \frac{1}{n} \sum_{i=1}^{n} z_{ik}^2 \tag{16}$$

Where $z_{ij}$ is the **score of observation** $i$ on component $k$. In other words, it is the **spread of data along PC**$k$.

Since the **PVE** is a proportion, the equation is pretty obvious:

$$\text{PVE}_k = \frac{\text{Var}_k}{\text{Var}_{\text{TOT}}} \tag{17}$$

It is the **percentage of total variance** explained by PC$k$. This tells us how informative PC$k$ is, and whether it's worth keeping or can be discarded.

In practice, we **often keep only the first few PCs**. For example, if we find that $PC1 + PC2$ explain 90% of the variance, we can keep only those two (dimensionality reduction).

## 1.7 Covariance vs. Correlation in PCA

In the previous sections, we discussed what PCA is and how we can use it. We know that PCA is based on the covariance matrix, that each principal component explains a portion of the total variance, and that PCA depends on the variance of the data. But "**what happens when the variables are on completely different scales?**" (Variable 1 is height, variable 2 is euros, variable 3 is just a count). If we apply PCA to the covariance matrix, the **variable with the largest variance will dominate the result**, even if it's not more important!

More formally, **PCA is based on the spread** (variance) **and co-movement** (covariance) **between variables**. But this depends heavily on the units of measurement and scales of the variables. And the problem is, should **we use the covariance matrix or the correlation matrix** to perform PCA? The answer depends on the scales of our variables.

### ⚠ Covariance Matrix: use with caution

The **covariance matrix measures how variables change together**, in ==absolute units==.

- ✔ This is fine only if **all variables are measured on comparable scales** (e.g., all in centimeters, or all in euros).

- ✘ If one variable has **much larger variance**, it will **dominate the PCA**, even if it's not the most important feature.

For example, suppose height (meters) has a variance of 0.01 and income (euros) has a variance of $10'000$. Income will control the direction of PC1 simply because its variance is greater, not because it's more meaningful.

### ✅ Solution: Standardize the Data

To prevent PCA from being biased toward large-scale variables, we **standardize** each variable:

$$X'_{ij} = \frac{X_{ij} - \bar{X}_j}{\text{SD}_j} \tag{18}$$

Where:

- $\bar{x}_j$ is the mean of variable $j$.

- $\text{SD}_j$ is the standard deviation of variable $j$.

This standardization ensures that the **mean is zero** and the **standard deviation is one**. Now **all variables are on the same scale** and PCA treats them equally.

✅ **Alternative: use the correlation matrix instead of the covariance matrix**

Another way to achieve this is to run PCA on the **correlation matrix**, which is simply the **covariance matrix of the standardized variables**. This work because:

$$\text{Corr}(X_j, X_k) = \frac{\text{Cov}(X_j, X_k)}{\text{SD}(X_j) \cdot \text{SD}(X_k)} \tag{19}$$

So if we **standardize the variables first, the covariance becomes correlation**.

| Case | Matrix to use | Why |
|---|---|---|
| All variables are on the same scale | Covariance matrix | Keeps units and variances as they are |
| Variables have different units/scales | Correlation matrix | Avoids bias toward large-variance variables |
| We standardized the variables | Covariance = correlation | Because standardization equalizes them |

Table 2: When to use Covariance or Correlation.

## 1.8   PCA via SVD - Computational Aspects

Instead of computing PCA by:

1. Building the covariance matrix $S = \dfrac{1}{n-1}X^T X$

2. Diagonalizing $S$ to get eigenvectors and eigenvalues

We can instead **perform PCA directly using Singular Value Decomposition (SVD)** on the **centered data matrix** $X$ itself.

### ▤ Singular Value Decomposition (SVD)

**Singular Value Decomposition (SVD)** method is a factorization of a matrix into three other matrices. If $X$ is the centered data matrix (size $n \times p$), then:

$$X = U\Sigma V^T \tag{20}$$

Where:

- **U**: matrix of **left singular vectors** (related to scores). It is orthogonal and $n \times n$.

- **Σ**: diagonal matrix of **singular values** $\Sigma_1, \Sigma_2, \ldots$

  - Non-negative real numbers on the diagonal (singular values)
  - Singular values sorted in descending order (from largest to smallest)
  - Number of eigenvalues guaranteed by the minimum between number of columns and number of rows: $\min(n, p)$.

- **V**: matrix of **right singular vectors**. It is orthogonal and $p \times p$. They are the main directions, so the loadings of PCA.

The eigenvalues of the covariance matrix can be calculated from the singular values $\Sigma_i$:

$$\lambda_i = \frac{\Sigma_i^2}{n-1}$$

This is because if we plug the SVD of $X$ into this formula, the eigen decomposition of $S$ naturally follows.

| Concept | Meaning/Role in PCA |
|---|---|
| $\mathbf{X} = \mathbf{U\Sigma V}^T$ | SVD decomposition of centered data matrix |
| $\mathbf{V}$ | Principal directions (same as eigenvectors of covariance) |
| $\Sigma_i$ | Singular values |
| $\lambda_i = \dfrac{\Sigma_i^2}{n-1}$ | Variance explained by each principal component |

Table 3: Summary table of SVD for PCA.

✅ **Advantages**

   ✔ **Numerically more stable**: works better when the data matrix is large or ill-conditioned.

   ✔ **No need to compute and store the covariance matrix**: we can work directly on $X$.

   ✔ **More efficient** when $p$ is large (many variables), especially in high-dimensional problems.

# 2   Clustering Methods

## 2.1   Introduction

In the previous section, we talked about PCA for compression or visualization. However, this topic falls under a type of technique called unsupervised learning.

- **Supervised Learning** is a type of machine learning where the algorithm is trained on a labeled dataset, meaning **each** training **example includes both the input data and the correct output**.

  The **goal** is to learn a function that maps inputs to outputs, in order to make predictions on new, unseen data. Typical tasks include:

  - Classification (e.g., spam detection, image recognition)
  - Regression (e.g., predicting house prices or credit scores)

- **Unsupervised Learning** is a type of machine learning where the algorithm is given **only input data without any labeled output**.

  The **objective** is to identify patterns, structures or groupings within the data. Common applications include:

  - Clustering (this section)
  - Dimensionality Reduction (e.g., PCA for compression or visualization, section 1.2, page 11)

In essence, while supervised learning is about *prediction*, unsupervised learning is about *discovery*.

### ❓ What is Clustering?

**Clustering** is the process of grouping a set of objects in such a way that **objects within the same group (cluster)** are **more similar** to each other than to those in other groups.

- No "true" labels are provided (unsupervised learning), the objective is to **uncover structure**.

- Often used in exploratory data analysis.

- The notion of "similarity" is fundamental and context-dependent (commonly based on **distance measures**).

The basic idea of clustering is:

1. **Minimize intra-cluster distances**: member of the **same cluster** should be **close to each other**.

2. **Maximize inter-cluster distances**: **different clusters** should be **well separated**.

---

**Example 1: Energy Consumption in Milan**

A practical example:

- Consider $n$ users, each described by their energy consumption across $p$ time slots.

- The task is to identify **groups of users with similar consumption patterns**, potentially revealing roles like *residences vs offices* or *daytime vs nighttime* usage.

- Since there is no predefined label, this is a classic unsupervised learning task.

This example encapsulates the **essence of clustering**, discovering structure in data that wasn't explicitly labeled.

---

### ❷ How Many Clusters?

A fundamental **challenge** in clustering is **deciding the number of clusters** ($k$). Visualizations show that data can often be reasonable clustered in 2, 4, or 6 groups, depending on the chosen definition or similarity.

The **clustering result depends on the similarity measure** (e.g., Euclidean distance), and the notion of a "true" cluster is often ambiguous. This ambiguity is a central theme in unsupervised learning: since there's no ground truth, **evaluating the "correctness" of a clustering is inherently difficult**.

### ≡ Types of Clustering

There are two main paradigms:

1. **Hierarchical Clustering**: builds a tree of clusters. Two main strategies:

   - **Agglomerative**: start from individual points and merge them.
   - **Divisive**: start from the whole dataset and split it recursively/

2. **Partitional Clustering** (also called flat clustering): divides data into $k$ non-overlapping groups. Each data point belongs to exactly one cluster.

Hierarchical methods are more interpretable, while partitional methods (e.g., k-means) are often faster and scalable.

## 2.2   Defining Similarity in Clustering

At the heart of clustering lies a simple yet crucial idea: "objects in the same cluster should be **more similar** to each other than to objects in other clusters". But how do we define "*similar*"? Clustering methods don't work in a vacuum, they **need a distance** (or similarity) **function to determine how close two data points are**. This choice directly affects the clustering result.

### ☰ Types of Distance Metrics

Given two data points $x = (x_1, x_2, \ldots, x_p)$ and $y = (y_1, y_2, \ldots, y_p)$, the distance between them can be computed in different ways, depending on the nature of the data and the goals of the analysis.

- **Euclidean Distance**

$$d_E\left(x, y\right) = \sqrt{\sum_{i=1}^{p}\left(x_i - y_i\right)^2} \tag{21}$$

  - Measures straight-line distance.
  - Works well when **features are on the same scale**.
  - Variants: squared euclidean and standardized euclidean (normalize variables before computing).

- **Manhattan Distance**

$$d_M\left(x, y\right) = \sum_{i=1}^{p}\left|x_i - y_i\right| \tag{22}$$

  Also called **L1 norm**, measures **grid-like distance** (think of navigating city blocks).

- **Chebyshev Distance**

$$d_C\left(x, y\right) = \max_i \left|x_i - y_i\right| \tag{23}$$

  Takes the **maximum absolute difference** across dimensions. Sensitive to the **worst-case** coordinate difference.

- **Cosine Similarity (Angle-based)**

$$\text{similarity}\left(x, y\right) = \cos\left(\theta\right) \tag{24}$$

  Based on the **angle between vectors**. The smaller the angle, the more similar the vectors. Common in **text mining** and **high-dimensional sparse data**.

- **Correlation-Based Distance**

$$d_R\left(x, y\right) = 1 - \text{corr}\left(x, y\right) \tag{25}$$

  Measures **shape similarity** and ignores magnitude; focuses on patter of variation.

- **Mahalanobis Distance**

$$d_M\left(x, y\right) = \sqrt{\left(x - y\right)^T \Sigma^{-1} \left(x - y\right)} \tag{26}$$

Accounts for **correlation between variables**. Useful when features have **different variances** or **covariances**.

- **Minkowski Distance**

$$d_p\left(x, y\right) = \left(\sum_{i=1}^{p} |x_i - y_i|^p\right)^{\frac{1}{p}} \tag{27}$$

General formula that generalizes Euclidean and Manhattan. If $p = 1$ is Manhattan, if $p = 2$ is Euclidean.

### ❓ Why Distance Matters

**Clustering doesn't "know" what matter, we tell it via distance**.

- The **choice of distance metric** can lead to **entirely different clusters**.

- Different applications require different definitions of similarity. For example, in customer segmentation, cosine similarity may be more appropriate than Euclidean distance, because we care about *purchasing trends*, not *absolute values*.

So instead of asking "*what's the best distance?*", ask: "***what kind of similarity is meaningful in our application?***"

## 2.3 Clustering Validity

Clustering can produce a result, a partition of our data, but **how do we know if it's good**? Unlike supervised learning, clustering lacks ground truth, so evaluating the quality of clustering is challenging. There are three main families of evaluation metrics, each answering a different kind of question: External Metrics, Internal Metrics, Relative Metrics.

### 2.3.1 External Metrics

**External metrics** assess the quality of a clustering by **comparing it to known class labels**. They answer the question: "how similar is the clustering result to the actual classification?". This is **possible only when ground truth labels[2] are available**, which is often the case in benchmarking or simulated data. However, these metrics are **not available in real-world applications where true labels are unknown**.

We usually **compare clusters** to classes using a **contingency table**:

|  | Class 1 | Class 2 | Class 3 | Total |
|---|---|---|---|---|
| Cluster 1 | $m_{11}$ | $m_{12}$ | $m_{13}$ | $m_1$ |
| Cluster 2 | $m_{21}$ | $m_{22}$ | $m_{23}$ | $m_2$ |
| Cluster 3 | $m_{31}$ | $m_{32}$ | $m_{33}$ | $m_3$ |
| Total | $c_1$ | $c_2$ | $c_3$ | $n$ |

- $m_{ij}$ number of points from class $j$ assigned to cluster $i$

- $m_i$ total points in cluster $i$

- $c_j$ total points in class $j$

- $n$ total number of points

From this, we compute **frequencies**:

$$p_{ij} = \frac{m_{ij}}{m_i} \tag{28}$$

---

[2]**Ground Truth Labels** are the true, correct categories or values assigned to data points. They represent the known answer. In clustering, ground truth refers to the real classification or grouping of our data, which is often manually annotated, observed from reality, or known from the context.

### 📋 Main External Metrics

- **Entropy Metrics**. Measures **class diversity within each cluster**.

  – A **pure cluster** (all elements from the same class) has entropy 0.
  – A **mixed cluster** hsa higher entropy (maximum when classes are equally mixed).

  The formula for **cluster** $i$:

  $$e_i = -\sum_{j=1}^{L} p_{ij} \log (p_{ij}) \tag{29}$$

  **Overall clustering entropy**:

  $$e = \sum_{i=1}^{K} \frac{m_i}{n} e_i \tag{30}$$

  Entropy decreases with better alignment. The **ideal value is zero**.

- **Purity Metrics**. Measures the **proportion of the dominant class in each cluster**. Like a "best guess" correctness. The formula for **cluster** $i$:

  $$p_i = \max_j (p_{ij}) \tag{31}$$

  **Overall clustering purity**:

  $$\text{purity} = \sum_{i=1}^{K} \frac{m_i}{n} p_i \tag{32}$$

  Purity increases with better alignment. The **ideal value is one**.

- **Precision Metrics** and **Recall Metrics**. These are adapted from classification metrics:

  – Precision (for cluster $i$, class $j$):

  $$\text{Prec}(i, j) = \frac{m_{ij}}{m_i} \tag{33}$$

  Among the points in cluster $i$, **how many truly belong to class $j$?**
  – Recall (for cluster $i$, class $j$):

  $$\text{Rec}(i, j) = \frac{m_{ij}}{c_j} \tag{34}$$

  Among all points from class $j$, **how many are captured by cluster $i$?**

- **F-Measure Metrics**. Combines precision and recall into a **single score** using the harmonic mean.

  $$F(i, j) = \frac{2 \cdot \text{Prec}(i, j) \cdot \text{Rec}(i, j)}{\text{Prec}(i, j) + \text{Rec}(i, j)} \tag{35}$$

  We can aggregate these across all clusters to get a **global F-score** for the clustering.

| Metric | Ideal | Interprets as... |
|--------|-------|------------------|
| Entropy | 0 | Purity of cluster (lower is better) |
| Purity | 1 | Dominance of single class (higher = better) |
| Precision | 1 | Cluster doesn't mix in wrong classes |
| Recall | 1 | Class is fully captured by a cluster |
| F-measure | 1 | Balanced precision and recall |

Table 4: Summary External Metrics.

### 2.3.2 Internal Metrics

When class labels (ground truth) are not available, which is the most common case in unsupervised learning, we need **Internal Metrics** to assess:

- How **cohesive** each cluster is (tightness of points within clusters).

- How **well-separated the clusters are from each other**.

These metrics measure **structural quality** based on distances between two points.

- **Sum of Squared Errors (SSE)**. Also called **Within-Cluster Sum of Squares (WCSS)**.

    ❓ **What is measures**
    * **How far** the **points** in each cluster are from their **cluster center** (or medoid)
    * **Lower SSE means tighter clusters**[3]

    📋 **Formula**

    $$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - c_i||^2 \tag{36}$$

    Where:
    * $C_i$ is cluster $i$
    * $c_i$ is the center of cluster $i$

    **Used to evaluate compactness**: a good clustering should have a small SSE.

---

[3]A tight cluster means that the data points inside the cluster are close to each other. They are packed together, not scattered.

- **Elbow Method**

  ❷ **What is measures**

  * Used with SSE to find the **optimal number of clusters** $K$.

  ✂ **Procedure**

  * Run clustering for various values of $K$ (e.g., 1 to 30)
  * Plot $SSE\,(K)$ vs $K$.
  * Find the "elbow" point: the value of $K$ where the SSE stops decreasing sharply.

  In other words, adding more clusters beyond the elbow gives **diminishing returns**.

- **Silhouette Coefficient**

  ❷ **What is measures**

  * **Combines cohesion and separation** into a single score for each point.
  * For each data point:
    · $a$: average distance to points in **same cluster**
    · $b$: average distance to points in **nearest other cluster**

  ▤ **Formula**

  $$s = \frac{b-a}{\max\,(a,b)} \qquad s \in [-1, 1] \tag{37}$$

  ✔ $+1$ well clustered
  * $0$ on the border
  ✘ $-1$ misclassified

  The **average silhouette score** across all points is **used to evaluate the whole clustering**.

- **Between-cluster Sum of Squares (BSS)**. Measures **cluster separation**:

  $$BSS = \sum_i m_i \, ||c_i - \bar{c}||^2 \tag{38}$$

  Where:

  - $m_i$: size of cluster $i$
  - $c_i$: center of cluster $i$
  - $\bar{c}$: global centroid

  A **good clustering** has **low WCSS** (or SSE) and **high BSS**.

### ⚠ Limitation of Internal Metrics

Most clustering algorithms **don't explicitly optimize internal metrics**. As a result, the **best-looking clustering under one metric may not be optimal in another**. For example, K-Means minimizes WCSS but may produce poor separation between clusters.

We can **design custom clustering algorithms that directly optimize an internal metric**, but this is <mark>not always practical or generalizable</mark>. A famous quote from Jain and Dubes (1988): "The validation of clustering structures is the most difficult and frustrating part of cluster analysis. Without a strong effort in this direction, clustering remains a black art accessible only to those true believers who have experience and great courage.". This reflects **how tricky and interpretation-heavy clustering evaluation can be**.

## 2.4   Hierarchical Clustering

**Hierarchical Clustering** is a **method of clustering that builds a hierarchy of clusters**, either by **merging** smaller clusters into larger ones (bottom-up) or **splitting** a large cluster into smaller ones (top-down).

- **Agglomerative Clustering (Bottom-Up)**

  1. Starts with **each data point as its own cluster**
  2. At each step:
     (a) Find the **two closest clusters**
     (b) **Merge** them into one
     (c) **Repeat** until there's one single cluster or a predefined number $k$

  This is the most common form of hierarchical clustering.

- **Divisive Clustering (Top-Down)**

  1. Starts with **all points in one large cluster**
  2. At each step:
     (a) **Split** one cluster into two
     (b) **Continue recursively** until each point is in its own cluster (or until $k$ is reached)

  This method is less commonly used due to higher complexity.

### ✂ Implementation Agglomerative Hierarchical Clustering

Hierarchical clustering works with a **distance matrix**, which **contains the distance between every pair of observations**. We don't need to specify $k$ beforehand, but we need a **stopping criterion** (e.g., stop when there are $k$ clusters, or use a threshold on distance).

1. Compute the **proximity matrix** (distances between all points)

2. Let each point be its **own cluster**

3. **Repeat**:

   (a) Find and **merge the two closest clusters**
   (b) **Update** the distance matrix

4. **Stop** when only one cluster remains (or another stopping condition is met)

The behavior of step 3, how we defined "closest clusters", depends on the **linkage method**.

## 🔗 Linkage Methods: How we Merge Clusters

The **linkage method** defines the distance between two clusters, based on the distances between points in those clusters.

1. **Single Linkage Method**

   ❓ **Distance**. It is the **minimum distance** between any two points (nearest neighbors)

   ✅ Can handle **non-globular shapes**.

   ⚠️ Sensitive to **noise and chaining effects**.

2. **Complete Linkage Method**

   ❓ **Distance**. It is the **maximum distance** between any two points (furthest points).

   ✅ Less sensitive to noise, but biased toward **spherical or compact clusters** (we mean that the method tends to favor or naturally produces clusters that have a specific shape or property).

3. **Average Linkage Method**

   ❓ **Distance**. It is the **average distance** between all points pairs.

   ✅ Uses **centroids** as reference and offers a **balance** between single and complete.

## ✅ Pros

   ✔️ Does not require predefining the number of clusters $k$.

   ✔️ **Produces a dendrogram**[4], a tree-like diagram showing how clusters are formed.

   ✔️ Can reveal structure at **multiple levels of granularity**.

## ❌ Limitations and Challenges

   ✖️ **Computational complexity**: distance matrix is $O(n^2)$, problematic for large datasets.

   ✖️ Once a merge/split is made, it **cannot be undone**.

   ✖️ There's **no direct optimization** of a global objective (unlike k-means minimizing SSE) .

   ✖️ **Results depend heavily on the linkage method used**, can lead to very different clusterings.

---

[4]The dendrogram is a tree where the leaves are individual data points and branches are merges between clusters. To choose $k$, "cut" the dendrogram at the level where the structure is stable (e.g., large vertical gaps suggest good splits).

## 2.5   K-Means

### 2.5.1   Introduction

**K-Means** is one of the most popular **clustering method**, specifically a type of **partitional clustering** (page 31). It's used to **group data into distinct clusters based on similarity**.

K-Means divides a dataset into $K$ groups, where:

- Each group is called a **cluster**.

- **Each cluster** has a **center** called **centroid**.

- Every data point belongs to the cluster with the **nearest centroid**.

### ❷ How does it work?

The K-means method can be divided into **five steps**:

1. **Choose** $K$ (the number of clusters).

2. **Randomly assign** each data point to a cluster.

3. **Compute the centroid** of each cluster (the mean of all its points).

4. **Reassign** each **point to the cluster** with the closest centroid.

5. **Repeat** steps 3 and 4 **until** the points **no longer change** clusters (convergence criterion).

The algorithm tries to **minimize the distance** between each point and the center of its cluster. This means we want the **clusters to be as compact** and **well-separated as possible**.

Note that this is only a simple, quick introduction to K-Means clustering. In the following chapters, we will delve deeply into each topic.

### 2.5.2   Definition

**K-Means** is a **unsupervised learning algorithm** used to partition a dataset into $K$ **distinct non-overlapping groups** called **clusters**, where each observation belongs to the cluster with the nearest mean (called the **centroid**).

### ◎ Main Goal

The goal of K-Means is to find a partition of the data that **minimizes the total within-cluster variation**, usually measured as the **sum of squared Euclidean distances** between each point and the centroid of its assigned cluster.

In other words, the K-means algorithm aims to **partition a dataset of $n$ observations into $K$ distinct clusters**:

- Each observation belongs to **one and only one** cluster.

- Each cluster has a **centroid**, i.e., a central point.

- Each point is assigned to the cluster with the **nearest centroid**.

- The goal is to **minimize the total distance** (squared Euclidean distance) from each point to its cluster's centroid.

We can express the previous goals in raw mathematical terms as follows:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \tag{39}$$

Where:

- $C_k$ is the **set of points** in the $k$-th cluster. Let $C_1, C_2, \ldots, C_K$ be the **index sets** of the clusters; these must form a **partition** of the dataset:

  - $C_1 \cup C_2 \cup \cdots \cup C_K = \{1, 2, \ldots, n\}$, every observation is included in some cluster.
  - $C_i \cap C_j = \emptyset$ for $i \neq j$, no observation can belong to two clusters at once.

  So, if the index $i \in C_k$, that means the $i$-th observation is assigned to the $k$-th cluster.

- $\boldsymbol{\mu}_k$ is the **centroid** (mean vector) of cluster $C_k$.

- $\|\mathbf{x}_i - \boldsymbol{\mu}_k\| = \|\cdot\|$ denotes **Euclidean distance**. It is the quantity that K-Means tries to minimize over all clusters.

| Concept | Meaning |
|---|---|
| **Clustering type** | Partitional (flat). |
| **Centroid** | Mean of points in a cluster. |
| **Assignment rule** | Assign each point to the **closest centroid**. |
| $K$ **(number of clusters)** | Must be **chosen in advance**. |
| **Objective function** | Minimize total within-cluster distance (e.g., sum of squared distances). |
| **Partition conditions** | Clusters are **disjoint and exhaustive** (no overlap, no omission). |

Table 5: K-Means summary

### ❷ How Many Clusters? (choosing $K$)

One of the main limitations of K-Means is that the number of clusters $K$ must be **specified in advance**. However, in real-world data, the true number of natural groupings is **usually unknown**. Choosing the wrong $K$ can lead to:

- **Underfitting**: too few clusters $\Rightarrow$ merging different groups

- **Overfitting** too many clusters $\Rightarrow$ splitting coherent groups

The common methods to choose $K$ are:

- **Elbow Method** looks at the **total within-cluster variation** (also called inertia or distortion), defined as:

$$W(K) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_K} \|\mathbf{x}_i - \mu_k\|^2 \tag{40}$$

  - $W(K)$ is the total within-cluster variability (it is explained in depth on page 49). It measures **how compact the clusters are**, the smaller $W(K)$, the tighter the points are around their centroids.

  - When $K = 1$, all points are in a single cluster, so $W(1)$ is very high. As $K$ increases, **clusters** become **smaller and tighter**, then $W(K)$ **decreases**. In the extreme case $K = n$, each point is its own cluster, then $W(n) = 0$.
  So, $W(K)$ **always decreases as $K$ increases**, but <u>not</u> linearly.

We compute Elbow Method for different values of $K$, and plot $W(K)$ vs $K$. The reason we plot the **within-cluster sum of squares $W(K)$** against the **number of clusters $K$** is to **visualize how the clustering quality improves** as we increase the number of clusters.

Plotting $W(K)$ against $K$ helps us **identify the point where adding more clusters gives diminishing returns**. It is called the "*elbow method*" because of the shape of the plot. When plotting $W(K)$ versus $K$, the curve usually drops sharply at first because adding more clusters

quickly improves clustering. After a certain point, however, adding more clusters only yields small improvements. The resulting plot often resembles a bent arm. The "*elbow*" is the point of maximum curvature, where the curve begins to flatten out. This point is important because **before the elbow**, we see **significant gains in reducing** within-cluster variance. **After the elbow**, however, the **gains are minimal**, indicating that increasing model complexity is not worthwhile.
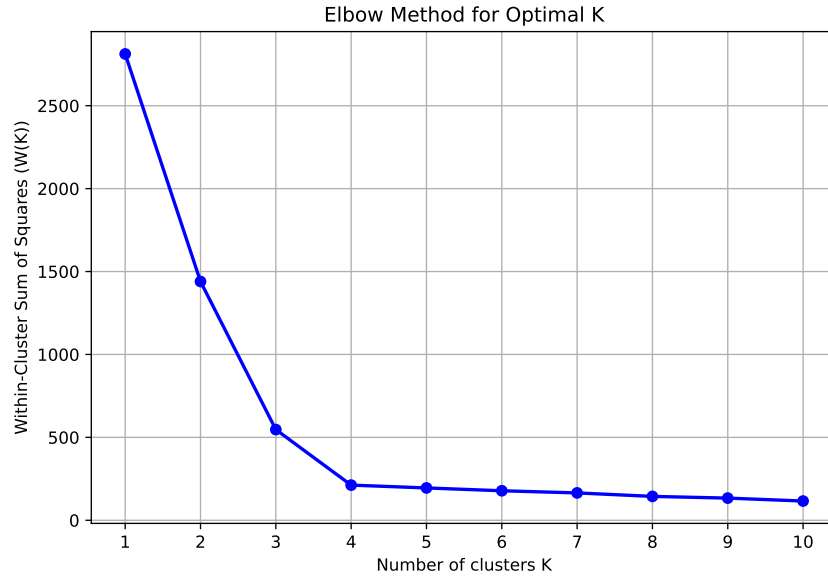


Figure 8: This is an example of the *elbow method* applied to K-means clustering. In the plot, we can see how the within-cluster sum of squares ($W(K)$) changes as the number of clusters $K$ increases from 1 to 10.

- When $K = 1$: $W(K)$ is very large (around 2500), since all points belong to a single, poorly-fitting cluster.

- From $K = 1$ to $K = 3$: $W(K)$ decreases rapidly, showing tat adding clusters significantly improves the clustering.

- From $K = 4$ onward: the curve starts to flatten, meaning adding more clusters gives only small improvements.

The elbow is visible at $K = 4$. This is where the rate slows down sharply. Adding more clusters doesn't significantly reduce $W(K)$. It balances clustering quality with model simplicity.

- **Silhouette Score** (or **Silhouette Coefficient**) evaluates **how well each point fits within its cluster** versus others. The Silhouette Coefficient for a point is:

$$s = \frac{b - a}{\max{(a, b)}} \tag{41}$$

  Where:

  - $a$ is the average distance to points in the same cluster

    − $b$ is the average distance to points in the nearest other cluster

Values close to 1 mean well-clustered points. We compute the average Silhouette score for all points and pick the $K$ that maximizes it.

- **Gap Statistics** (too advanced) compares the total within-cluster variation for the real data to that for randomly generated data with no structure. A large "gap" means the real data forms more distinct clusters. Choose the smallest $K$ for which the gap is maximal or within 1 standard deviation of the maximum.

---

**Example 2: K-Means**

Below is a simple run of the K-means algorithm on a random dataset.

- Iteration 0 - **Initialization**
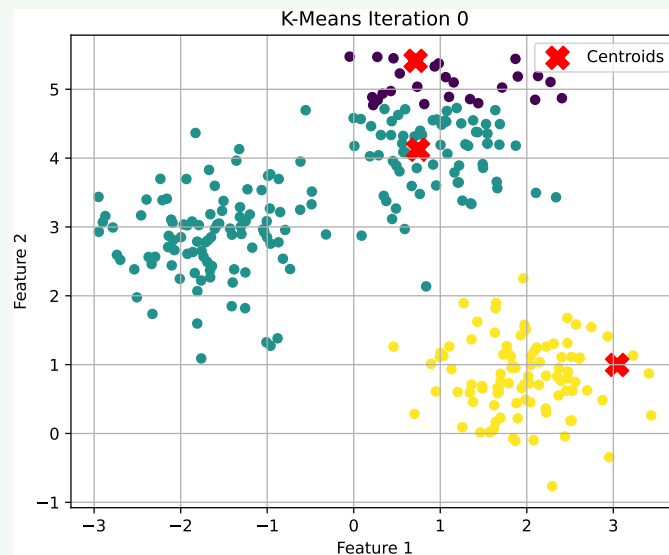


This is the starting point of the K-Means algorithm. **Three centroids are randomly placed in the feature space**. At this point, no data points are assigned to clusters yet, or all are assumed to be uncolored/unclustered. The positions of the centroids will strongly influence how the algorithm proceeds.

The goal here is to start with some guesses. The next step will use these centroids to form the initial clusters.

- Iteration 1 - **First Assignment and Update**

---
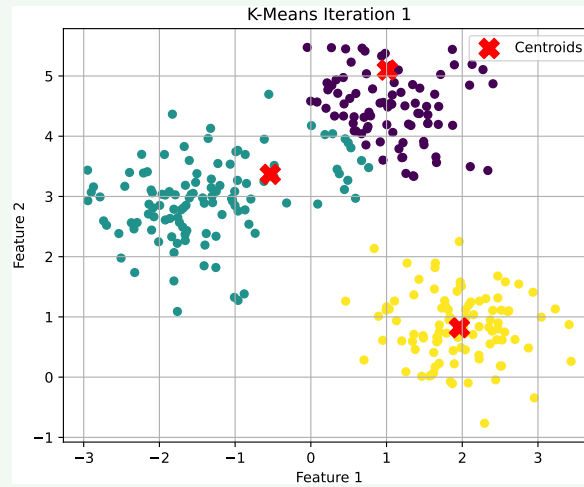
Each data point is assigned to the closest centroid, forming the first version of the clusters. New centroids are computed by taking the average of the points in each cluster. We can already see structure forming in the data, as points begin grouping around centroids.

This step is the first real clustering, and centroids begin to move toward dense regions of data.

- Iteration 2 - **Re-Assignment and Refinement**
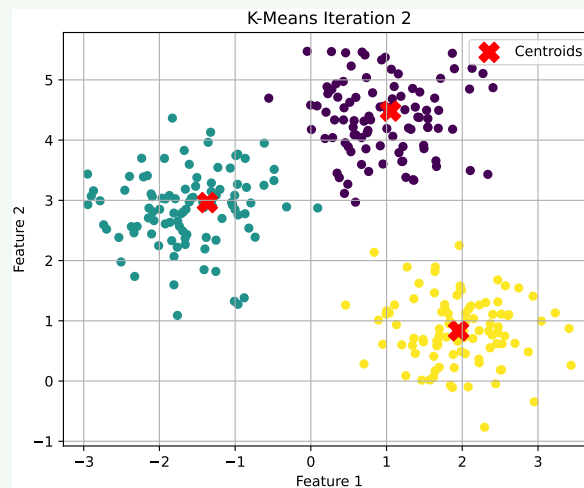


Clusters are recomputed based on updated centroids. Many points remain in the same clusters, but some may shift to a new cluster if a centroid has moved. Centroids continue moving closer to the center of their respective groups.

The algorithm is now refining the clusters and reducing the total distance from points to centroids.

- Iteration 3 - **Further Convergence**
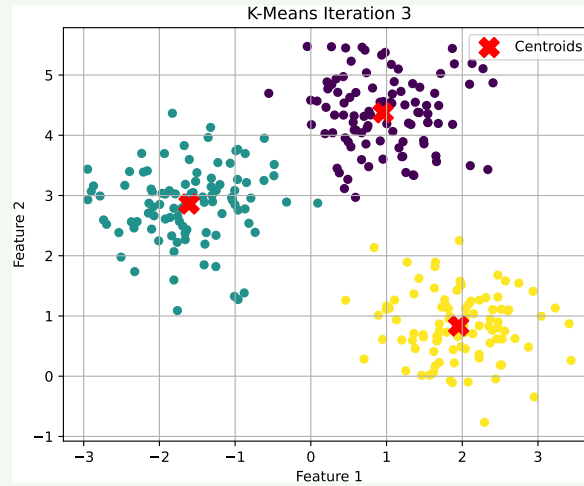


At iteration 3, the K-Means algorithm reached convergence. The centroids no longer moved, and no points changed cluster.This means:

- The algorithm has found a locally optimal solution.

- Further iterations would not improve or change the clustering.

- The final configuration is considered the result of the algorithm.

In practice, this is how K-Means stops: it checks whether the centroids remain unchanged, and if so, it terminates automatically.

### 2.5.3 Initialization Issues in K-Means

### ❷ Why initialization matters

The K-Means algorithm starts by choosing $K$ **initial centroids**. These **starting positions are crucial** because K-Means builds the entire clustering process based on them.

Since K-Means is an **iterative algorithm**, it gradually improves the clusters from where it begins. However, if it starts from a bad configuration (i.e. bad initial centroids), it can easily get stuck in a **local minimum**, a clustering solution that isn't optimal but looks good from that starting point.

This means that:

- **Different initial centroids** can lead to **different final results**.

- Some clusters ma be poorly formed or even empty.

- The quality of the final clustering can vary, even on the same data.

### ⚠ What can go wrong

- **Centroids too close**: if initial centroids are placed near each other, they might all end up capturing the same group of points, leaving other areas underrepresented.

- **Unbalanced clusters**: a poor start can lead to clusters with very different sizes or shapes, even if the data has nicely separated groups.

- **Empty clusters**: it's possible that some centroids end up with no points assigned, which breaks the algorithm's assumptions.

### ✔ How to improve initialization (conceptually)

To avoid these problems, two common strategies are used:

1. **Multiple random initializations**. Instead of relying on just one random start, K-Means is **run several times with different initial centroids**. The **result with the lowest total distance is selected**. This increases the change of finding a good clustering.

2. **Smart initialization**. Instead of choosing all centroids randomly, a better approach is to choose the first centroid randomly, then pick the next one in a way that they are **as far apart as possible**. This increases the chances of covering different regions of data from the beginning.

   This smarter approach significantly reduces the risk of bad clustering and makes the algorithm more stable and reliable.

### 2.5.4   K-Means as an Optimization Problem

Given:

- A set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of $n$ data points in $\mathbb{R}^d$

- A chosen number of clusters $K$

The goal of K-Means is to **partition the data into $K$ clusters**:

$$C = \{C_1, C_2, \ldots, C_K\}$$

So that a measure of **within-cluster variability** is **minimized**.

### ❷ What is Within-Cluster Variability?

In simple terms:

- **Cluster**: a group of points that are similar (close together).

- **Variability**: how spread out the points are.

- **Within-Cluster Variability** how spread out the points are **inside each cluster**.

So within-cluster variability measures **how close the points in the same cluster are to each other**. We have low variability when the points are tightly packed and high variability when the points are spread out.

### ❷ Okay, but why do we need to minimize within-cluster variability?

Because a **good cluster** should contain **points that are close together**, not randomly scattered. So K-Means algorithm tries to form clusters where the **internal "spread" is as small as possible**.

### ❷ Objective Function: what are we minimizing?

For each cluster $C_k$, define its **centroid $\mathbf{c}_k$** as the **mean of the points in the cluster**. The within-cluster variability is measured by the following:

$$W\left(C_k\right) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{d} \left(x_{ij} - x_{i'j}\right)^2 \tag{42}$$

- Each observation is a point:

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in \mathbb{R}^d$$

That means $\mathbf{x}_i$ is a vector with $d$ features.

- $\displaystyle\sum_{i,i' \in C_k}$ means we are looking at **all pairs of points** $i$ and $i'$ inside the same cluster $C_k$.

- $\sum_{j=1}^{d} (x_{ij} - x_{i'j})^2$, for each feature $j$, we compute the **squared distance** between point $i$ and point $i'$.

- $\sum_{i,i' \in C_k} \sum_{j=1}^{d} (x_{ij} - x_{i'j})^2$ measures the **total squared distance** between **all point pairs inside the cluster**.

- $\frac{1}{|C_k|}$, we divide by the number of points to get an **average distance**.

So $W(C_k)$ is the **average squared distance between all pairs of points inside cluster** $C_k$.

### 🏎️ Optimization Criterion

The full K-Means objective is to **find the clustering** that **minimizes the total within-cluster variability**:

$$\min_{C_1,\dots,C_K} \sum_{k=1}^{K} W(C_k) \tag{43}$$

This means we are searching for the best possible partition $\{C_1, \dots, C_K\}$ that results in clusters where **points are as close as possible to each other**, i.e., the clusters are compact.

In other words, we want **every cluster to be compact**. So we add up the within-cluster variability for all clusters. We ==minimize the problem while keeping it as compact as possible==.

### ✅ Interpretation

A **good clustering** has **low within-cluster variability**: points in the same cluster are close together. Also, K-Means solves an optimization problem where it tries to find such a clustering by minimizing a specific cost function.

### 2.5.5   Algorithm

The K-Means algorithm is an iterative method that alternates between assigning data points to clusters and updating the cluster centroids, with the goal of minimizing within-cluster variability.

0. **Choose the number of clusters $K$**. Before running the algorithm, we must decide the number of clusters $K$. Each cluster will have:

   - A **set of points** $C_1, \ldots, C_K$
   - A **centroid** $\mathbf{v}_1, \ldots, \mathbf{v}_K$
   - A **partition matrix** $\Gamma = [\gamma_{ij}]$ where:

   $$\gamma_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_j \in C_i \\ 0 & \textbf{otherwise} \end{cases}$$

1. **Initialization**. Randomly assign each observation to one of the $K$ clusters. Then compute the **initial centroids** for each cluster:

   $$\mathbf{v}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$$

   These are the **mean vectors** of the assigned points.

2. **Assignment**. For each point $\mathbf{x}_i$, compute the distance to all centroids, and assign it to the cluster with the **closest centroid**. This updates the **partition matrix** $\Gamma$.

3. **Update Centroids**. After the new assignments, **recompute the centroid** of each cluster using the updated points:

   $$\mathbf{v}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$$

**Repeat Step 2 and Step 3** until:

- ✔ No points change cluster (==convergence==), or

- ✔ A ==stopping criterion== is met (e.g., max iterations)

📗 **Properties of the algorithm**

✔ **Good**

- – The algorithm **always converges** (it stabilizes after a finite number of iterations)
- – Simple and fast

✖ **Bad**

- – It does **not always find the global optimum**
- – The **results depends** heavily on the **initialization**

In practice, K-means clustering is **typically run multiple times with different random initializations**, and the **result with the lowest cost is kept**.

### 2.5.6   Limitations

While K-Means is simple and efficient, it has several important limitations:

1. **We must choose the number of clusters $K$ in advance**. Unlike **hierarchical clustering**, which builds a full tree of clusterings, K-Means requires we to **fix $K$ before the algorithm starts**.

   ✖ Choosing the wrong $K$ can lead to poor results.

   ✖ There's no universal rule for selecting $K$; methods like the Elbow or Silhouette must be used.

2. **Assumes clusters are globular (spherical) in shape**. K-Means tends to form **round-shaped clusters** due to the use of **Euclidean distance**.

   ✖ It **struggles with elongated, irregular, or non-convex shapes**.

   ✖ Works best when clusters are **well-separated, equally sized, and dense**.

3. **Fails with clusters of different sizes or densities**. If one cluster is very dense and another is sparse, K-Means might:

   ✖ Split one true cluster into several pieces

   ✖ Merge multiple small clusters into one

   This happens because Euclidean distance doesn't account for cluster **spread or shape**.

4. **Sensitive to outliers**. K-Means uses **centroids (means)** to define clusters. Since the **mean is sensitive to extreme values**, an outlier can significantly shift a centroid and distort the clustering.

   As an alternative, we could use **K-Medoids**, which is **based on medians rather than means** and is more robust to outliers.

5. **Sensitive to feature scaling and representation**. Because K-Means relies on distance, the algorithm is:

   ✖ Affected by **unscaled or poorly represented data**

   ✖ Sometimes improved by **dimensionality reduction** (e.g., PCA)

## 2.6   Gaussian Mixture Models (GMMs)

### 2.6.1   Introduction

A **Gaussian Mixture Model (GMM)** is a **probabilistic model** that assumes the data is generated from a **mixture of several Gaussian distributions** (also called normal distributions), each representing a different **latent group or cluster**.

**Each data point** is not assigned to one specific cluster (as we have seen in the K-means algorithm), but is instead modeled as **having a probability of belonging to each cluster**.

❓ **Why is it called a "*mixture*"?**

Because the model represents the **overall data distribution** as a **weighted sum of several Gaussian components**. Mathematically, the probability density function is:

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

Where:

- $K$ is the number of **Gaussian components** (clusters)

- $\pi_k$ is the **mixing coefficient** of component $k$ (like a *weight*), with $\sum \pi_k = 1$

- $\mathcal{N}\left(x \mid \mu_k, \Sigma_k\right)$ is the **multivariate Gaussian distribution** with:

    - Mean $\mu_k$
    - Covariance matrix $\Sigma_k$

A more detailed explanation can be found on page 60.

---

**Remark: What is the Multivariate Gaussian Distribution?**

❓ **What is a 1D Gaussian (Normal) Distribution?**

The **1D Gaussian** (also called **Normal Distribution**) is the classic bell curve and it tells us how likely we are to get values near the **mean $\mu$**, and how unlikely values far from the mean are. The spread is controlled by the **standard deviation $\sigma$**.

In other words, it describes a **continuous** variable whose values are **most likely to occur near a central point**, and **less likely the farther we move away**.

The **Probability Density Function (PDF)** of a 1D Gaussian is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{44}$$

- $x$ is the value of the random variable.

- $\mu$ is the **mean**, the center of the distribution.

- $\sigma$ is the **standard deviation**, it controls the "spread" of the curve.

The bell-shaped curve is symmetric around the mean $\mu$. Most values fall within:

- $\mu \pm \sigma$: $\approx 68\%$

- $\mu \pm 2\sigma$: $\approx 95\%$

- $\mu \pm 3\sigma$: $\approx 99.7\%$

For example, if we have a 1D Gaussian with $\mu = 0$, $\sigma = 1$, it is called the **standard normal distribution**, where the curve peaks at 0 and spreads out smoothly in both directions.

### ❷ What is a 2D Gaussian Distribution?

The **2D Gaussian** (or **Bivariate Normal Distribution**) is the natural extension of the 1D Gaussian to **two dimensions**. It tells us how likely a point $(x, y)$ is to appear in a **2D space**, just like how the 1D Gaussian tells us about single values on a line.

A **2D Gaussian distribution** is completely defined by:

1. **Mean Vector** $\boldsymbol{\mu} \in \mathbb{R}^2$

   This is the **center** (or peak) of the distribution.

   $$\boldsymbol{\mu} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \tag{45}$$

   It tells us the **average position** of the points, the location where the distribution is highest.

2. **Covariance Matrix** $\Sigma \in \mathbb{R}^{2 \times 2}$

   This tells us how the data is **spread out and oriented** in 2D space.

   $$\Sigma = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} \tag{46}$$

   The diagonal values control the **spread** along each axis, and the off-diagonal values ($\rho$) represent **correlation** between $x$ and $y$.

   - $\sigma_x^2$, $\sigma_y^2$ is the variances along the X and Y axes.
   - $\rho$ is the correlation between $x$ and $y$.
     - If $\rho = 0$: the axes are independent.
     - If $\rho > 0$: the ellipse tilts upward (positive correlation).
     - If $\rho < 0$: the ellipse tilts downward (negative correlation).

In short, a 2D Gaussian is a "bell-shaped hill" in 3D space, centered at $\boldsymbol{\mu}$, with shape and tilt determined by $\Sigma$.
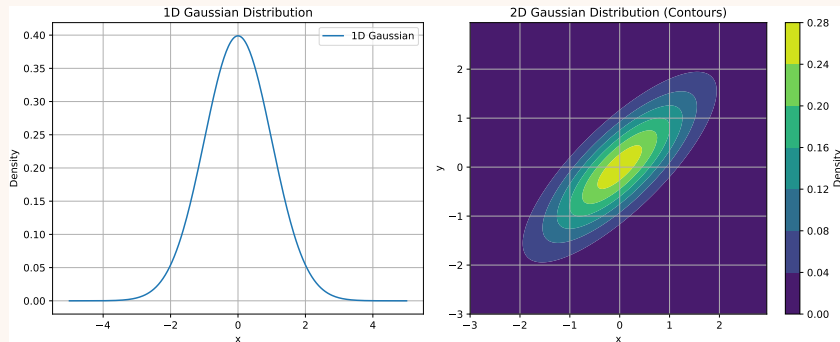
Figure 9: 1D Gaussian and 2D Gaussian (as Contours). In the **1D plot** (left), we see the classic **bell-shaped curve**. This shows the probability density of a variable $x$ under a Gaussian distribution centered at $\mu = 0$ with standard deviation $\sigma = 1$.

- The peak occurs at the mean.

- The density decreases symmetrically as we move away from the mean.

- Most of the probability mass lies within $\pm 1$ or $\pm 2$ standard deviations.

This curve describes how **likely** different values of $x$ are under the distribution. The second plot (right) shows a **2D Gaussian distribution using contour lines**. This represents a Gaussian defined over two variables, $x$ and $y$.

- The **concentric ellipses** indicate regions of equal probability density (like topographic maps).

- The ellipses are **tiled**, showing that the variables $x$ and $y$ are **positively correlated**.

- The closer to the center, the higher the density.

- A 2D Gaussian distribution models probabilities over two variables $\mathbf{x} = (x, y)$. At each point $(x, y)$, the distribution gives a **density value**, that is, how likely it is that a data point would occur at that location. This means the 2D Gaussian is actually a function from 2D to 1D:

$$f(x, y) = \text{density}$$

So even though it's defined in 2D, it **outputs a single number** (the density).

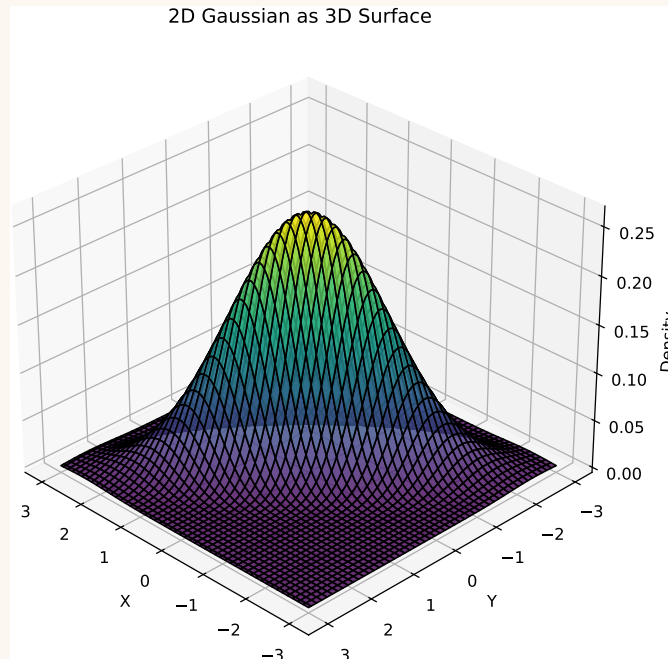This view helps visualize the **shape and orientation** of the distribution in 2D space.

Figure 10: A 3D plot showing the same 2D Gaussian as a surface in three dimensions.

- The horizontal plane is the $x$-$y$ space.

- The vertical axis shows the **density** at each point.

- The result is a **bell-shaped hill**, centered at the mean.

- The ridge direction and slope reflect the **correlation and spread** between the two variables.

We use $x$ and $y$ as the horizontal plane (input space), and $z$-axis as **density value** $f(x, y)$. Now, we are plotting the **full shape of the probability surface**: a bell-shaped hill where the height indicates the likelihood of a given point. We add a **third axis to represent the function output**, the density. While **2D contours** are useful for seeing **direction**, **correlation**, and **structure**, a **3D surface** provides a **better sense of the bell shape**.

### ✅ What is the Multivariate Gaussian Distribution?

The **Multivariate Gaussian Distribution** is simply a **generalization of the 1D and 2D Gaussian** to **any number of dimensions**. If we have a random vector $\mathbf{x} \in \mathbb{R}^d$, then: a **multivariate Gaussian** models the probability of observing different values of that vector, assuming that all components jointly follow a Gaussian distribution.

## √x̄ Formal Definition

A random vector $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ follows a multivariate normal distribution with:

- **Mean vector**: $\boldsymbol{\mu} \in \mathbb{R}^d$

- **Covariance matrix**: $\Sigma \in \mathbb{R}^{d \times d}$

We write this as:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \tag{47}$$

Its **probability density function** is:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \cdot \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{48}$$

- $\boldsymbol{\mu}$ is the **center** (expected value) of the cloud of points.

- $\Sigma$ is the **shape and orientation** of that cloud

  - The diagonal entries $\sigma_{ii}$ control how spread out the data is in each direction.
  - The off-diagonal entries $\sigma_{ij}$ represent **correlation** between different variables.

So a multivariate Gaussian forms an **ellipsoidal distribution** in high-dimensional space.

### ❷ Why GMM uses Multivariate Gaussian Distribution?

In Gaussian Mixture Models, each component is a **multivariate Gaussian**:

- It models a cluster in **multidimensional space**.

- The full GMM is a **weighted sum** of these multivariate Gaussians.

This allows GMM to **model complex, overlapping, and anisotropic (non-spherical) clusters** much better than K-Means.

### ▤ Relation to K-Means

K-Means is a **special case of GMM**:

- K-Means assigns each point to **only one cluster**, called **Hard Clustering**.

- GMM assigns each point a **probability of belonging** to each cluster, called **Soft Clustering**.

In fact, __if we assume__ that:

- All clusters have **equal, spherical** covariance.

- Each point is assigned to the **closes cluster center**.

Then *GMM reduces to K-Means.*

### ❷ But why should we use GMM instead of K-means?

Four reasons:

1. **Soft Clustering vs Hard Clustering**

   - **Hard Assignment**: K-Means assigns each point to **only one cluster**.
   - **Soft Assignment**: GMM assign each point a probability of belonging to each cluster.

   GMM is useful when clusters overlap, we're uncertain or want probabilistic interpretation, or we want to model real-world uncertainty (e.g., "60% likely to be in Cluster 1").

2. **Flexible Cluster Shapes**

   K-Means assumes clusters are **spherical and equally sized**. Instead, GMM allows:

   - Elliptical shapes;
   - Different sizes and orientations;
   - Correlations between features.

   This is because GMM uses **covariance matrices**, whereas K-Means just uses Euclidean distance.

3. **Statistical Foundation**

   K-Means is an algorithm that minimizes distance. Instead, GMM is a **probabilistic model** with a well-defined likelihood function. Then, we can estimate the uncertainty and use model selection to determine the number of clusters.

4. **K-Means is a Special Case of GMM**

   If we assume:

   (a) All clusters have **equal and diagonal covariance matrices** (no correlation);
   (b) Use hard assignments.

   Then GMM becomes K-Means. So GMM is a **more general, more powerful model**.

However, GMM is not always the best method to consider. It depends on the data and the final goal:

- Use **K-Means** when **clusters** are clearly **separated**, **round**, and **fast results** are needed.

- Use **GMM** when we want a **flexible**, **probabilistic**, and **realistic model** of how our data is generated.

| Feature | K-Means | GMM |
| --- | --- | --- |
| Assignment | Hard | Soft (probabilistic) |
| Cluster shape | Spherical | Elliptical, flexible |
| Model type | Geometric | Probabilistic (generative) |
| Speed | Fast | Slower (EM algorithm) |
| Interpretability | Simple centroids | Full statistical model |

Table 6: Trade-offs between K-Means and GMM.

### 2.6.2  Mathematical Foundation

Even though we have already introduced the GMM formula, we provide a thorough explanation here.

### ❓ What is a Mixture Model?

A **mixture model** assumes that our data was generated from **multiple probability distributions**, not just one. For GMMs, we assume: each data point $\mathbf{x}_i \in \mathbb{R}^d$ was **generated by one of several multivariate Gaussian distributions**. But we don't know:

- *Which point came from which component.*

- *What the component parameters are.*

### 📋 GMM Definition

The probability density function of a Gaussian Mixture Model with $K$ components is:

$$p\left(\mathbf{x}\right) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k\right) \tag{49}$$

Where:

- $\mathbf{x} \in \mathbb{R}^d$ is the observed data point.

- $\pi_k$ is the **mixing coefficient** for component $k$, with $\sum_{k=1}^{K} \pi_k = 1$, and $\pi_k \geq 0$.

- $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$ is the **multivariate Gaussian density** of the $k$-th component, with:

  - Mean $\boldsymbol{\mu}_k \in \mathbb{R}^d$
  - Covariance matrix $\Sigma_k \in \mathbb{R}^{d \times d}$

  This is the **probability density** of the Multivariate Gaussian Distribution, also called the **likelihood of observing the point x**, **given the parameters of component** $k$ (mean $\mu$ and covariance $\Sigma$).

  For example, imagine we're standing at a point $\mathbf{x}$ in space, and asking: *how likely is it that a point like this came from Gaussian cluster k?* The answer is:

  $$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$$

  The value is **higher** if $\mathbf{x}$ is **closer to the center** of cluster $k$, and **lower** if it's farther away.

The meaning of each term is as follows:

| Symbol | Meaning |
| --- | --- |
| $\pi_k$ | Prior probability of choosing cluster $k$. |
| $\boldsymbol{\mu}_k$ | Mean (center) of the $k$-th Gaussian component. |
| $\Sigma_k$ | Covariance (shape and orientation) of the $k$-th component. |
| $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$ | Likelihood of $\mathbf{x}$ under the $k$-th Gaussian. |
| $p(\mathbf{x})$ | Overall probability of observing $\mathbf{x}$ under the full model. |

To generate a data point $\mathbf{x}$, first choose a component $k$ with probability $\pi_k$, then draw $\mathbf{x}$ from the Gaussian $\mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$. This is why GMM is a generative model, **it models how data could have been generated**.

### 2.6.3   Responsibilities

Unlike **K-Means**, where each point is assigned to exactly **one cluster** (hard clustering), **GMM** performs **soft clustering**: each point has a probability of belonging to every cluster. These probabilities reflect *how likely* it is that a point came from each component of the mixture.

### ❷ Why do we need to introduce Responsibilities in GMM?

⚠ We have a **problem** at the moment. When using a Gaussian Mixture Model, we assume that each data point $\mathbf{x}_i$ was generated by **one of the** $K$ **Gaussian components**. But, we **don't know which one**. That information is **hidden** or **latent**. We only observe:

- The data point $\mathbf{x}_i$

- The parameters of the Gaussian $(\pi_k, \boldsymbol{\mu}_k, \Sigma_k)$

We don't observe which component $k$ generated $\mathbf{x}_i$.

*I still don't understand what the problem is.* In a Gaussian Mixture Model, we assume that each data point $\mathbf{x}_i$ is generated by **one of the** $K$ **Gaussian components**. So the true process that generates each point is:

1. Randomly choose a component $k$ with probability $\pi_k$.

2. Generate a point $\mathbf{x}_i$ from the Gaussian $\mathcal{N}(\mu_k, \Sigma_k)$

But we don't know which Gaussian generated each point and this is the core issue:

- We **observe only the data points** $\mathbf{x}_1, \dots, \mathbf{x}_n$.

- We **do not observe the "label"** of which component (cluster) generated each point.

That **label is missing**! This is problematic because, if we want to **train a model** (i.e., estimate the $\pi_k$, $\mu_k$, and $\Sigma_k$), we would **ideally want to group the data by component**. However, we can't do that because we don't know which point belongs to which component.

*Okay, but if we're using the GMM formula, then we know everything about each cluster that produces each point.* That's a good point, but the practice is slightly different. In the real world, especially in machine learning, we don't know the origin of each point because we only have the observed data. So, we are trying to **infer** the underlying **clusters**, **without knowing** which one generated each point.

---

**Example 3: analogy of the GMM problem**

Imagine a factory that has 3 machines (A, B, C) making cookies. Each machine:

- Adds a different amount of sugar and chocolate

---

- Sometimes makes similar-looking cookies, but not exactly the same

- Has a different production rate (e.g., Machine A makes 50% of all cookies, B makes 30%, C makes 20%)

Now, someone gives us a box of mixed cookies and says: "*These were made using machines A, B, and C, but I'm not telling you which cookie came from which machine.*" We taste each cookie and write down how sweet and chocolatey it is. That's our data.

⚠ **The Problem.** We don't know:

- Which machine made each cookie.

- How sweet/chocolatey each machine's recipe really is.

- How often each machine is used.

In GMM terms:

- We have the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (cookie features).

- But the label "*this came from machine A*", is hidden.

✅ **The Solution.** Since we can't know for sure where each cookie came from, we say: "*Let me estimate how likely it is that this cookie came from each machine.*". So for Cookie #1, we might say:

- 70% Machine A

- 20% Machine B

- 10% Machine C

This set of numbers is the responsibility vector for Cookie #1. We do this for every cookie. Now, even though we don't know the true labels, we have soft guesses, and that's enough to:

- Estimate what each machine's recipe looks like (mean and variance).

- Estimate how many cookies each machine probably made (mixing proportions).

The core idea is that we need "responsibilities" because the cluster label is missing. Rather than guessing the labels directly, we estimate the probability that each point belongs to each cluster. These probabilities are called "responsibilities".

✅ **Solution: Responsibilities = Posterior Probabilities**

We come back to GMM. We assume that:

1. Each point $\mathbf{x}_i$ was generated by **one of $K$ Gaussians**.

2. But we **don't observe** which one.

So we introduce a **latent variable** $z_i \in \{1, \ldots, K\}$ that **represents the unknown cluster** of $\mathbf{x}_i$ (hidden cluster assignments).

We introduce the idea of a **latent variable** $z_i$, which says: "let's pretend each point $\mathbf{x}_i$ has a hidden label $z_i \in \{1, \ldots, K\}$". But since we don't observe $z_i$, we'll estimate the **probability** that it takes each value.

A **Latent Variable** is a variable that:

1. **Exists conceptually**, and

2. **Affects the data**, but

3. Is **not directly observed** in our dataset

In other words, a latent variable is something we don't see, but it explains patterns in the things we do see.

> **Example 4: continuation of the example on page **
>
> Let's go back to our cookie story.
>
> - We **see the cookie** $\Rightarrow$ that's observed data.
>
> - We **don't see the machine** that made it $\Rightarrow$ that's a **latent variable**.
>
> The **machine label** (e.g. A, B, or C) is a latent variable $z$ because:
>
> 1. It determines the cookie's sweetness/chocolate level.
>
> 2. But we aren't told what it is, we must infer it.

Since we cannot observe $z_i$ directly, we can **estimate the probability** that $z_i = k$ for each component $k$. This estimate is called **Responsibility** and is mathematically expressed as follows:

$$\gamma_{ik} = P\left(z_i = k \mid \mathbf{x}_i\right) \tag{50}$$

We are trying to *approximate* the unknown values of $z_i$. This enables us to **learn from incomplete data** without needing the true labels.

| Latent Variable | Responsibility |
|---|---|
| Missing truth | Out best guess of that truth. |

The responsibility is computed using **Bayes' Rule**:

$$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k\right)}{\displaystyle\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \Sigma_j\right)} \tag{51}$$

In simple terms, we have:

- The numerator is: how well component $k$ explains point $\mathbf{x}_i$, weighted by how likely that component is in general.

- The denominator ensures that the total probability sums to 1 (normalization across all components).

The **main idea** behind GMM is very simple. Rather than stating that "point $i$ is in cluster 2", GMM says that point $i$ is 80% likely to be in cluster 2, 15% likely to be in cluster 1, and 5% likely to be in cluster 3. These values are the **responsibilities** $\gamma_{ik}$. The word *responsibility* reflects that **each component is partially responsible** for explaining each point.

### 2.6.4  Expectation-Maximization (EM) Algorithm

The **Expectation-Maximization (EM) Algorithm** is how we **train** a Gaussian Mixture Model (GMM). Meaning, how we **estimate** the unknown parameters:

- Mixing coefficients $\pi_k$

- Means $\boldsymbol{\mu}_k$

- Covariances $\Sigma_k$

We use the EM algorithm **due to the unknown cluster labels**, $z_i$. The data is incomplete, so we cannot group the points by component. The EM algorithm usually works when some variables are latent (hidden).

### ⚒ How EM Works (Big Picture)

EM is an **iterative algorithm** with two main steps:

1. **E-step (Expectation)**. In this step, we **compute responsibilities**:

$$\gamma_{ik} = P\left(z_i = k \mid \mathbf{x}_i\right) = \frac{\pi_k \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k\right)}{\displaystyle\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \Sigma_j\right)}$$

   This is the **expected value of the latent variable** $z_i$, given the current parameter estimates. In simpler terms: we update our guess of how much each cluster $k$ is responsible for each point $\mathbf{x}_i$.

2. **M-step (Maximization)**. In this step, we **update the model parameters** using the responsibilities:

   (a) **Update weights** $\pi_k$:

$$\pi_k = \frac{1}{n} \cdot \sum_{i=1}^{n} \gamma_{ik}$$

   (b) **Update means** $\boldsymbol{\mu}_k$:

$$\boldsymbol{\mu}_k = \frac{\displaystyle\sum_{i=1}^{n} \gamma_{ik} \cdot \mathbf{x}_i}{\displaystyle\sum_{i=1}^{n} \gamma_{ik}}$$

   (c) **Update covariances** $\Sigma_k$:

$$\Sigma_k = \frac{\displaystyle\sum_{i=1}^{n} \gamma_{ik} \cdot \left(\mathbf{x}_i - \boldsymbol{\mu}_k\right) \cdot \left(\mathbf{x}_i - \boldsymbol{\mu}_k\right)^T}{\displaystyle\sum_{i=1}^{n} \gamma_{ik}}$$

We're **maximizing the likelihood** of the data under the current soft assignments.

We alternate E and M steps **until the log-likelihood stops improving significantly**, or we **reach a maximum number of iterations**. This gives us the best estimate of the parameters, despite not knowing the cluster labels.

### ❷ What EM is really doing

Each iteration:

- Uses the **E-step** to "*guess*" hidden information (soft labels).

- Uses the **M-step** to "*retrain*" the model based on those guesses.

That's why it's called **Expectation-Maximization**:

- We **expect** the <mark>missing values</mark>.

- Then **maximize** the <mark>likelihood given that expectation</mark>.

### 2.6.5   Comparison between GMM and K-Means

Even though both Gaussian Mixture Models (GMM) and K-Means are used for **clustering**, they differ significantly in their assumptions, outputs, and flexibility.

- **Conceptual Difference**

| Concept | K-Means | GMM |
|---|---|---|
| Type | Geometric algorithm | Probabilistic generative model |
| Goal | Minimize distances to centroids | Maximize likelihood of data under Gaussian model |
| Output | Hard assignments (1 cluster) | Soft assignments (probabilities per cluster) |

- **Shape and Structure**

| Feature | K-Means | GMM |
|---|---|---|
| Cluster Shape | Spherical (equal radius) | Elliptical (any orientation/size via covariance) |
| Covariance Info | Not modeled | Fully modeled via $\Sigma_k$ for each cluster |
| Equal size | Assumes clusters are similar in size | Allows different size and shape per cluster |

- **Assignment Method**

    - **K-Means** assigns each point to the **nearest centroid** (based on Euclidean distance).
    - **GMM** assigns each point a **probability** of belonging to each cluster (based on likelihood).

    This makes GMM better for:

    - Overlapping clusters.
    - Uncertain or fuzzy boundaries.
    - Data with correlated features.

- **Optimization Strategy**

    - **K-Means** minimizes **within-cluster sum of squared distances**.
    - **GMM** maximizes the **log-likelihood** of the observed data via the **EM algorithm**.

- **Computational Cost**

| Aspect | K-Means | GMM |
|---|---|---|
| Speed | Faster | Slower (EM is iterative) |
| Convergence | Guaranteed | Also guaranteed, but may find local optima |
| Initialization | Important | Even more sensitive to initialization |

- **Summary Table**

| Feature | K-Means | GMM |
|---|---|---|
| Assignments | Hard | Soft (probabilities) |
| Cluster shape | Spherical | Elliptical (via covariance) |
| Uses probability model? | ✘ No | ✔ Yes |
| Optimization objective | Distance | Log-likelihood |
| Suitable for overlapping? | ✘ No | ✔ Yes |

- **When to Use What?**

| Use Case | Choose |
|---|---|
| We need fast, simple partitioning | K-Means |
| We want a statistical model or density info | GMM |
| Clusters overlap or vary in shape | GMM |
| We have very large datasets | K-Means (or approximate GMM) |

# 3 Discriminant Analysis

## 3.1 From Unsupervised to Supervised

So far, we've been working with **GMMs** and **clustering**, which are **unsupervised** methods. This meaning:

- We had no labels.

- We tried to discover patterns (e.g., clusters) **just from the data**.

Now we're moving to **Supervised Learning** where:

- Each observation $\mathbf{x}_i$ **comes with a label** $y_i \in \{1, \dots, K\}$

- We **want to learn a function** that can classify new inputs based on past labeled data.

### ❓ Why This Transition Is Natural (GMM → LDA)

Let's revisit how we modeled things in GMM. In GMM, we assumed:

$$\mathbf{x} \sim \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(\mu_k, \Sigma_k)$$

But we **didn't know** which $k$ each $\mathbf{x}$ came from, hence, EM algorithm. In Supervised Learning (like Linear Discriminant Analysis, LDA), we **do know** which class $k$ each $\mathbf{x}$ comes from. This lets us directly estimate the parameters $\pi_k, \mu_k, \Sigma_k$ from the **labeled data**. Thus:

| GMM (Unsupervised) | LDA/QDA (Supervised) |
|---|---|
| No labels | Labels $y_i \in \{1, \dots, K\}$ |
| Estimate hidden assignments (EM) | Directly estimate class parameters |
| Responsibilities: soft membership | Class probabilities from Bayes rule |
| Soft clustering | Hard classification |

| Term | Meaning |
|---|---|
| **Latent Variable** | In GMM, the unknown class membership $Z$; in supervised learning, it's known. |
| **Generative Model** | A model for how data is generated: first sample label $y \sim \pi_k$, then $\mathbf{x} \sim \mathcal{N}(\mu_k, \Sigma_k)$. |
| **Supervised Learning** | Learn a classifier from labeled data — i.e., a function $f(\mathbf{x}) = \hat{y}$. |
| **Discriminant Analysis** | Use Bayes' rule with class-conditional distributions to assign classes. |

### ✖ **Key Formulation Shift**

In GMM (unsupervised), we care about:

$$\mathbb{P}(\text{class } = k \mid \mathbf{x}) = \frac{\pi_k \cdot \phi\left(\mathbf{x} \mid \mu_k, \Sigma_k\right)}{\sum\limits_{j=1}^{K} \pi_j \cdot \phi\left(\mathbf{x} \mid \mu_j, \Sigma_j\right)}$$

Now in **supervised learning**, we do the same, but:

- We estimate $\mu_k, \Sigma_k$ **per class** from labeled data.

- Then use Bayes' rule to classify a new point $\mathbf{x}$.

This is the basis for LDA and QDA (Quadratic Discriminant Analysis).

We're not throwing away the GMM mindset, we're now using it **with supervision**:

- Same assumptions: Gaussian class-conditional densities.

- But now, we **know the true class**, so we don't need EM.

This is why **LDA** is sometimes called the **supervised version of GMM with equal covariances**.

## 3.2  Introduction to Supervised Learning

**Supervised Learning** is about learning a mapping from inputs to outputs using **labeled data**. We're given a dataset:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\} \tag{52}$$

Where:

- $\mathbf{x}_i \in \mathbb{R}^p$: vector of features (e.g., age, salary, pixel values, etc.).

- $y_i \in \{1, 2, \ldots, K\}$: label.

Our goal is to **learn a function**:

$$f(\mathbf{x}) = \hat{y} \tag{53}$$

That can predict the correct $y$ for a new input $\mathbf{x}$.

### ≡ Types of Supervised Learning

| Problem Type | Output $y$ | Examples | Algorithms |
|---|---|---|---|
| Classification | Categorical/Discrete | Tumor type | LDA, QDA, logistic reg |
| Regression | Real-valued | Temperature | Linear regression |

### ⚖ Supervised vs. Unsupervised

| | Unsupervised | Supervised |
|---|---|---|
| Input | $\mathbf{x}_i$ | $(\mathbf{x}_i, y_i)$ |
| Goal | Discover structure (clusters) | Predict outcome $y$ from $\mathbf{x}$ |
| Methods | GMM, K-Means | LDA, QDA |

So in GMM, we tried to **infer the labels**. In Supervised Learning, the labels are **given**, so the task becomes more about **learning rules**.

### ❷ What is Discriminant Analysis, and how is it linked to Supervised Learning?

Because **Discriminant Analysis** is one of the **oldest and most classic forms of supervised learning**, specifically for **classification** tasks.

In Supervised Learning, we want to **learn a rule that predicts the class $y$ given an input x**. Discriminant Analysis **does exactly that**, by:

1. Using the labeled data $(\mathbf{x}_i, y_i)$

2. Estimating how each class looks in feature space, with a **probability model** for $\mathbb{P}(\mathbf{x} \mid y = k)$

3. Applying **Bayes' rule** to invert that: estimate $\mathbb{P}(y = k \mid \mathbf{x})$

4. Choosing the class with the highest probability

So we use the label $y_i$ to:

- Group the data by class;

- Estimate a distribution for each class;

- Then **discriminate** new points by which class distribution they're most likely to belong to.

🗒 **More in-depth. Discriminant Analysis** is a method to **classify observations into categories** (like "spam" vs "not spam", or species A vs species B) by:

1. **Modeling how each class generates data**, using a probability distribution (usually multivariate normal).

2. Then using **Bayes' Rule** to compute the probability that a new point belong to each class.

3. Finally, assigning the new point to the **most probable class**.

The word *discriminant* refers to the idea of **finding rules or boundaries that discriminate** between classes based on the features $\mathbf{x}$. So:

- LDA: uses linear boundaries

- QDA: uses quadratic boundaries

These are both types of discriminant functions.

Discriminant Analysis **is called generative because it is a generative model** that assumes we can model:

- $\mathbb{P}(\mathbf{x} \mid y = k)$: the distribution of features **given** the class.

- $\mathbb{P}(y = k)$: the prior probability of each class.

Using Bayes' theorem:

$$\mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x} \mid y = k) \cdot \mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})}$$

This is discriminant analysis, we **use these class-conditional densities to make predictions**.

### Definition 1: Generative Model

A **Generative Model** models **how the data is generated**, i.e., it learns the **joint distribution**:

$$\mathbb{P}(\mathbf{x}, y)$$

From which we can **generate new data**, or compute the conditional:

$$\mathbb{P}(y \mid \mathbf{x})$$

More specifically, in classification tasks, a generative model models:

$$\mathbb{P}(\mathbf{x} \mid y = k) \quad \text{(class-conditional density)}$$

And:

$$\mathbb{P}(y = k) \quad \text{(class prior)}$$

Then it uses **Bayes' Rule** to compute:

$$\mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x} \mid y = k) \cdot \mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})}$$

#### Example 1: LDA as a Generative Model

LDA assumes:

- Each class $k$ has a Gaussian distribution:

$$\mathbf{x} \mid y = k \sim \mathcal{N}(\mu_k, \Sigma)$$

- We learn:
  - $\mu_k$, the mean of class $k$
  - $\Sigma$, shared covariance matrix
  - $\pi_k$, prior probability of class $k$

With those, we can:

- **Generate data**: sample a class $y \sim \pi_k$, then sample $\mathbf{x} \sim \mathcal{N}(\mu_k, \Sigma)$

- **Classify new points** using **Bayes' Rule**

### ⚒ Practical Importance

- **Data**: we often have real-world datasets where class labels are known (e.g., in medicine, finance, spam detection).

- **Goal**: we want to *classify* new observations as accurately as possible.

- **Approach**: fit models like LDA, logistic regression, or modern ML classifiers (SVM, Trees, etc.).

## 3.3   Linear Discriminant Analysis (LDA)

**Linear Discriminant Analysis (LDA)** is a **supervised classification method** that models the probability distribution of each class as a multivariate **Gaussian with the same covariance matrix**, and uses **Bayes' rule** to assign a new observation to the class with the **highest posterior probability**.

Because it assumes equal covariances across classes, the resulting **decision boundaries are linear**, hence the name *Linear* Discriminant Analysis.

### ❷ What LDA Tries to Do

We're given data from multiple classes (say $K = 2$ or more), and we want to **assign a new point x to the most likely class**. LDA does this by:

1. Modelling how each class generates data (generative model).

2. Using Bayes' rule to compute $\mathbb{P}\left(y = k \mid \mathbf{x}\right)$.

3. Assigning $\mathbf{x}$ to the class with the highest probability.

---

> **Example 2:  LDA**
>
> Suppose we want to classify fruit by weight and color:
>
> - Oranges: heavy, orange.
>
> - Lemons: light, yellow.
>
> LDA will:
>
> - Estimate the **mean** and **spread** (covariance) of orange and lemon data.
>
> - Find a **line** that best separates the two, that's the linear discriminant.

---

### 📑 Assumptions Behind LDA

1. **The data in each class looks like a cloud of points shaped like a Gaussian**. Imagine we have two groups of data:

   - Group A, orange points
   - Group B, blue points

   Each group is made of points in 2D (like height and weight). We **assume** that in each group, the points:

   (a) Are **spread out in a bell-shaped way**.
   (b) Cluster around a mean (center).
   (c) Are more dense in the middle, less dense on the sides.

That's called a **multivariate normal distribution** (like a 2D Gaussian blob).

❷ **Why?** Because this makes the math nice. We can use formulas to describe where the data is most likely to appear in each class.

2. **All the groups are spread out in the same way**. This means:

   - Even though Group A and Group B may be **centered in different places** (==different means==),
   - They are **equally wide, equally tilted, and equally stretched** (==same covariance==).

   Two round clouds of points, one at the left, one at the right, then same shape, just different centers.

   ❷ **Why?** Because this gives us **linear boundaries**. Meaning, the separator between groups will be a **straight line** (or a plane in higher dimensions). That's why it's called *Linear* Discriminant Analysis.

3. **We roughly know how frequent each class is**. This is called the **prior probability**: if we know that 70% of our training data is oranges and 30% lemons, we should take that into account when classifying.

   ❷ **Why?** Because we don't want to treat all classes as equally likely if they aren't. Priors help us adjust for that.

### 🗐 Derivation via Bayes Rule

To really understand how LDA works, not just *what it does* but *why it works*, we need to know that LDA classifies a point $\mathbf{x}$ by computing "*which class is most likely to have generated this point?*". And the way to compute that is:

- Estimate how likely $\mathbf{x}$ is *under each class* (the likelihood).

- Multiply it by how common that class is (the prior).

- Then use Bayes' Rule to get the final answer: the **posterior probability**.

✅ **Goal.** Specifically, we want to **assign a new observation**, $\mathbf{x}$, to one of $K$ classes by computing the following:

$$\mathbb{P}(y = k \mid \mathbf{x}) \quad \text{(posterior probability)} \tag{54}$$

It is the **Posterior Probability**, which answers the question, "*given an observation $\mathbf{x}$, what is the probability that it belongs to class $k$?*" It's called the **posterior** probability because it's what we know **after** seeing the data $\mathbf{x}$. In other words, this is the **probability that x belongs to class** $k$, given the observed features.

1. **Use Bayes' Rule**. Bayes' Rule helps us to **compute the posterior**:

$$\mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x} \mid y = k) \cdot \mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})}$$

Where:

- $P(y = k \mid \mathbf{x})$ is the posterior, what we want to compute.
- $P(\mathbf{x} \mid y = k)$ is the <mark>likelihood</mark>, how likely $\mathbf{x}$ is if it comes from class $k$.
- $P(y = k)$ is the <mark>prior</mark>, how likely class $k$ is overall.
- $P(\mathbf{x})$ is the <mark>evidence</mark>, same for all classes, just a normalizing constant.

2. **What LDA Assumes**. LDA **assumes** that:

   (a) Each class $k$ generates data using a **Gaussian distribution**:

   $$\mathbf{x} \mid y = k \sim \mathcal{N}(\mu_k, \Sigma)$$

   (b) All classes share the same covariance $\Sigma$.

   (c) The prior $P(y = k) = \pi_k$ is known or estimated from the data.

3. **Simplify the Formula**. Because $\mathbb{P}(\mathbf{x})$ is the same for every class (it doesn't depend on $k$), we can **ignore it when choosing the best class**. So the posterior is **proportional** to:

   $$\mathbb{P}(y = k \mid \mathbf{x}) \propto \pi_k \cdot \phi(\mathbf{x} \mid \mu_k, \Sigma)$$

   Where:

   - $P(y = k \mid \mathbf{x})$ is the posterior, what we want to compute.
   - $\propto$ is the proportional symbol.
   - $\phi(\mathbf{x} \mid \mu_k, \Sigma)$ is the Gaussian probability density for class $k$.

   To make this easier to work with, we **take the logarithm** of this expression (log doesn't change which class is biggest). This gives the **Discriminant Function**:

   $$\delta_k(\mathbf{x}) = \log(\pi_k) - \frac{1}{2}\,\mu_k^T\,\Sigma^{-1}\,\mu_k + \mathbf{x}^T\,\Sigma^{-1}\,\mu_k \tag{55}$$

   Then we classify $\mathbf{x}$ to the class $k$ with the **highest** $\delta_k(\mathbf{x})$.

The final rule is that we need to **assign x to the class $k$ for which $\delta_k(\mathbf{x})$ is the largest**. This is the <mark>LDA decision rule</mark>.

### ❓ What is the Decision Rule?

The **LDA Decision Rule** assigns a new observation $\mathbf{x}$ to the class $k$ with the **highest posterior probability**:

$$\hat{y} = \arg\max_k \mathbb{P}(y = k \mid \mathbf{x}) \tag{56}$$

But since **computing** that posterior **directly is messy**, we used **Bayes' Rule to rewrite it** as:

$$\mathbb{P}(y = k \mid \mathbf{x}) \propto \pi_k \cdot \phi(\mathbf{x}; \mu_k, \Sigma)$$

So instead, we define a **discriminant function**:

$$\delta_k(\mathbf{x}) = \log(\pi_k) - \frac{1}{2}\,\mu_k^T\,\Sigma^{-1}\,\mu_k + \mathbf{x}^T\,\Sigma^{-1}\,\mu_k$$

Then the decision rule becomes:

$$\hat{y} = \arg\max_k \delta_k(\mathbf{x}) \tag{57}$$

### ❷ Geometry: Why It's Called Linear

LDA is called Linear Discriminant Analysis because the decision boundary it creates between classes is a **straight line** (in 2D), or a **hyperplane** (in higher dimensions).

### ❷ What's a decision boundary? It's the set of points $\mathbf{x}$ where:

$$\delta_k(\mathbf{x}) = \delta_l(\mathbf{x})$$

i.e., where the classifier is **undecided** between class $k$ and class $l$. This boundary tells us: *"here, both classes are equally likely, it's the tipping point between them."*.

### ▤ Remark: LDA Discriminant Function. The discriminant function:

$$\delta_k(\mathbf{x}) = \log(\pi_k) - \frac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + \mathbf{x}^T \Sigma^{-1} \mu_k$$

It's **linear in x**, there's no square, no exponent, just a dot product of $\mathbf{x}$ with some vector. So if we compare two classes $k$ and $l$, their difference:

$$\delta_k(\mathbf{x}) - \delta_l(\mathbf{x})$$

Is also **linear in x**.

### Example 3: Plot of a 2D LDA

Let's say we have 2 features: height (cm), weight (kg). Then the LDA boundary between class A and class B will look like a straight line in the 2D plane:
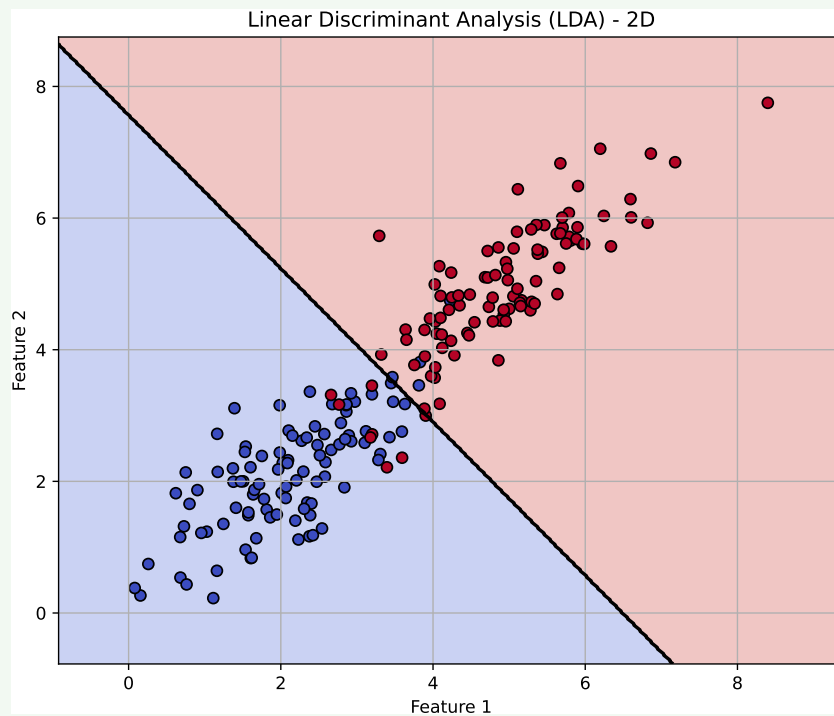


All the points on the blue side belong to class A, and all the points on the other side belong to class B.

However, LDA support any number of classes $K \geq 2$. With 2 classes, the decision boundary is a single line; with 3 or more classes, LDA creates multiple regions, each with its own boundary between class pairs.

LDA Decision Boundaries (3 Classes in 2D)



LDA Decision Boundaries (4 Classes in 2D)

LDA Decision Hyperplane (3D)
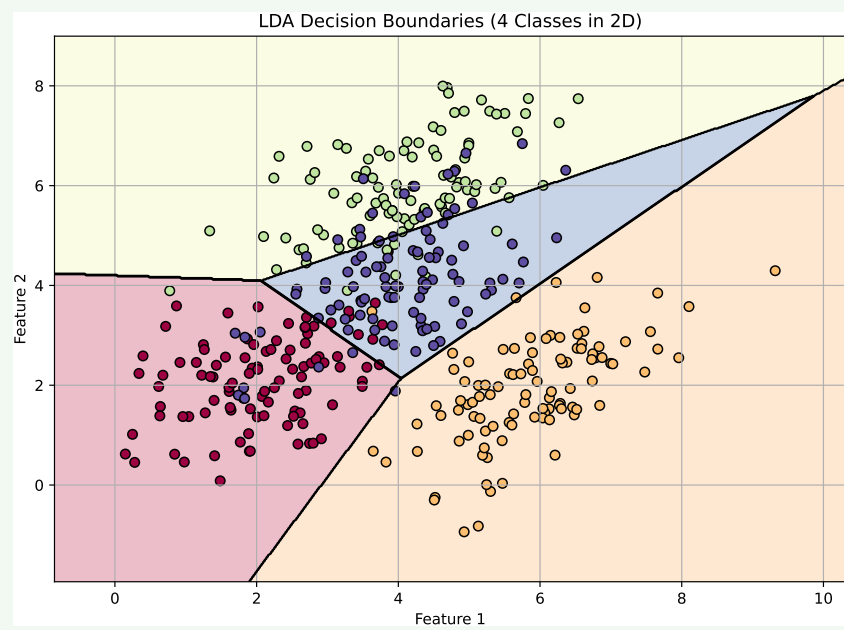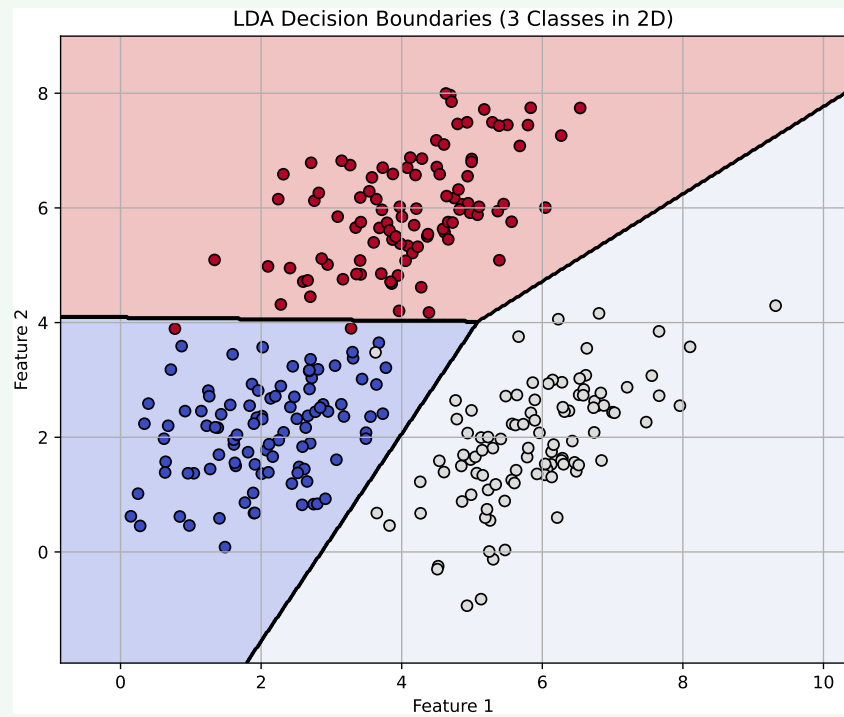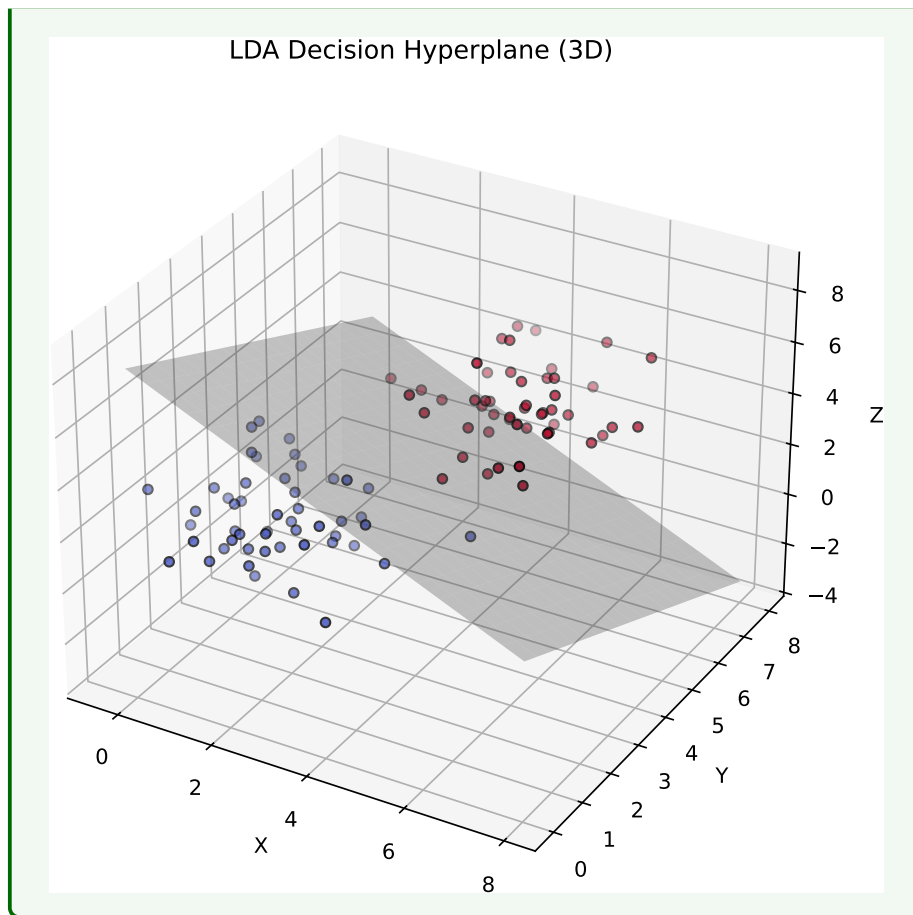
## 3.4   Quadratic Discriminant Analysis (QDA)

**Quadratic Discriminant Analysis (QDA)** is a **supervised classification method** that models each class as a multivariate Gaussian distribution **with its own covariance matrix**.

It uses **Bayes' Rule** to classify new observations by computing posterior probabilities, like LDA, but allows **more flexible class shapes**.

### 📗 Intuition

QDA is just like LDA, **but more flexible**:

- In **LDA**, all classes share the same shape (same covariance, then same spread, same orientation).

- In **QDA**, each class has its **own shape and orientation** in feature space.

This means we can model curved boundaries and non-symmetric class regions. But the **price is more parameters**, and **more chance of overfitting** on small datasets.

### ⚠ What is Overfitting?

**Overfitting** occurs when a **model learns** not just the **true patterns** in the data, **but also the random noise**. It's like memorizing answer for an exam instead of understanding the subject; we do well on the questions we saw before (training data), but fail on new ones (test data).

Imagine fitting a curve to data points:

| Fit Type | Description |
|---|---|
| **Underfitting** | The model is too simple (e.g., a straight line when the pattern is curved), it misses important structure. |
| **Good Fit** | The model captures the true pattern without being too complex. |
| **Overfitting** | The model bends and twists to go exactly through every training point, even if those points contain random noise. |

Table 7: Fitting type.

Our model is overfitting when we observe the following:

- **Training error**: very *low*

- **Test error**: *high*

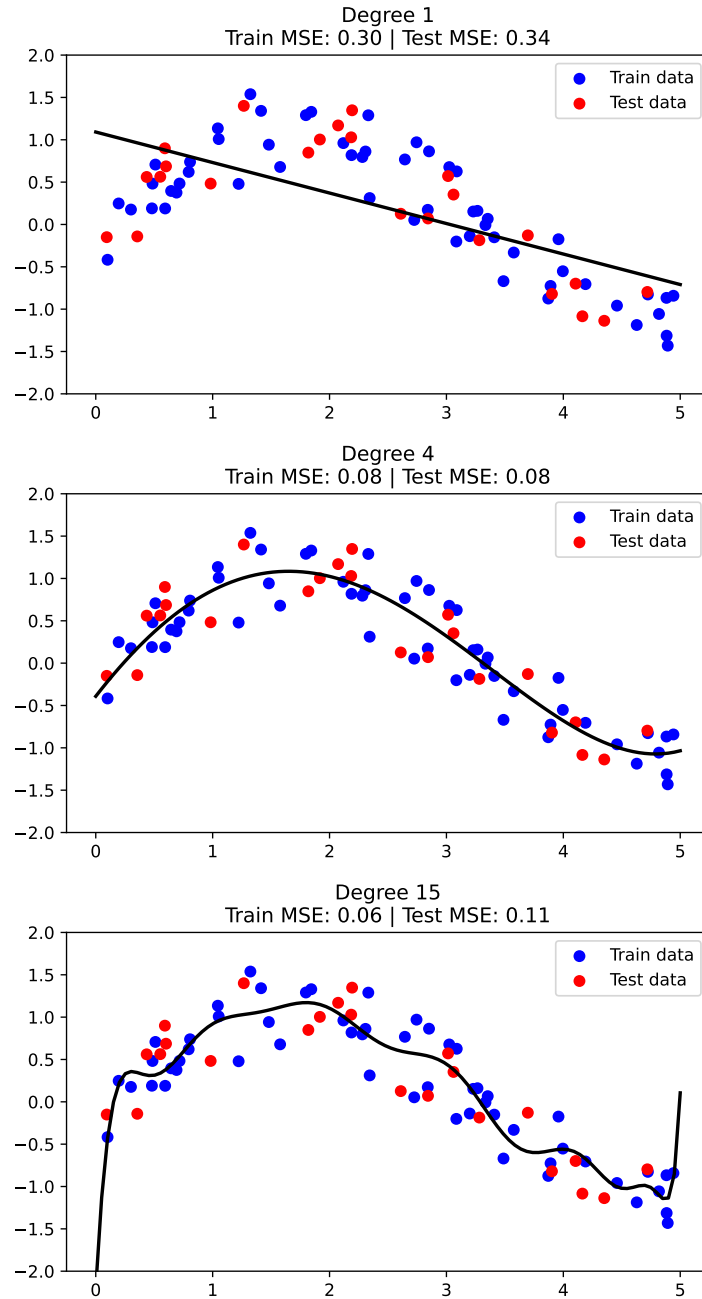The model performs **too well** on the training set because it's **memorized it**, not generalized it.

Figure 11: An example of overfitting. This figure shows polynomial regression models with increasing complexity. Although polynomial regression itself hasn't been explained, the purpose of this figure is to illustrate the effect of overfitting as model complexity increases (degree 1 underfits, too simple, degree 4 fits well, good generalization, degree 15 overfits, captures noise).

## ⚖️ How does QDA differ from LDA?

- **Covariance Matrix**

| LDA | QDA |
|---|---|
| Shared across all classes, $\Sigma$ | Unique per class, $\Sigma_k$ for each class |

Covariance Matrix.

- **Decision Boundary**

| LDA | QDA |
|---|---|
| **Linear** (straight lines / hyperplanes) | **Quadratic** (curved lines / surfaces) |

Decision Boundary.

- **Model Complexity**

| LDA | QDA |
|---|---|
| Simpler, fewer parameters to estimate | More complex, more parameters (one full covariance per class) |

Model Complexity.

- **Bias vs. Variance**

| LDA | QDA |
|---|---|
| Lower variance, higher bias (more stable) | Lower bias, higher variance (risk of overfitting) |

Bias vs. Variance.

- **Shape of Class Regions**

| LDA | QDA |
|---|---|
| All classes modeled with the **same** **shape & orientation** | Each class can have a **different** **shape & orientation** |

Shape of Class Regions.

- **Interpretability**

| LDA | QDA |
| --- | --- |
| Easier to interpret decision boundaries | Harder to visualize, especially in higher dimensions |

Interpretability.

- **Performance with Few Data**

| LDA | QDA |
| --- | --- |
| Works well, robust | Needs more data to estimate separate covariances reliably |

Performance with Few Data.

## ❓ What's the key difference?

LDA and QDA both model each class as a multivariate Gaussian, but the key difference is:

- **LDA assumes all classes share the same covariance matrix**. The data clouds (ellipses) all have the same shape and orientation.

- **QDA allows each class to have its own covariance matrix**. The data clouds (ellipses) can have different shapes, sizes, and orientations.

## ❓ What this means in practice

| Concept | LDA | QDA |
| --- | --- | --- |
| **Boundary shape** | Straight lines (linear) | Curved lines (quadratic) |
| **Flexibility** | Less, all classes "look similar" | More, each class can "look unique" |
| **Math form** | Linear discriminant function | Quadratic discriminant function |
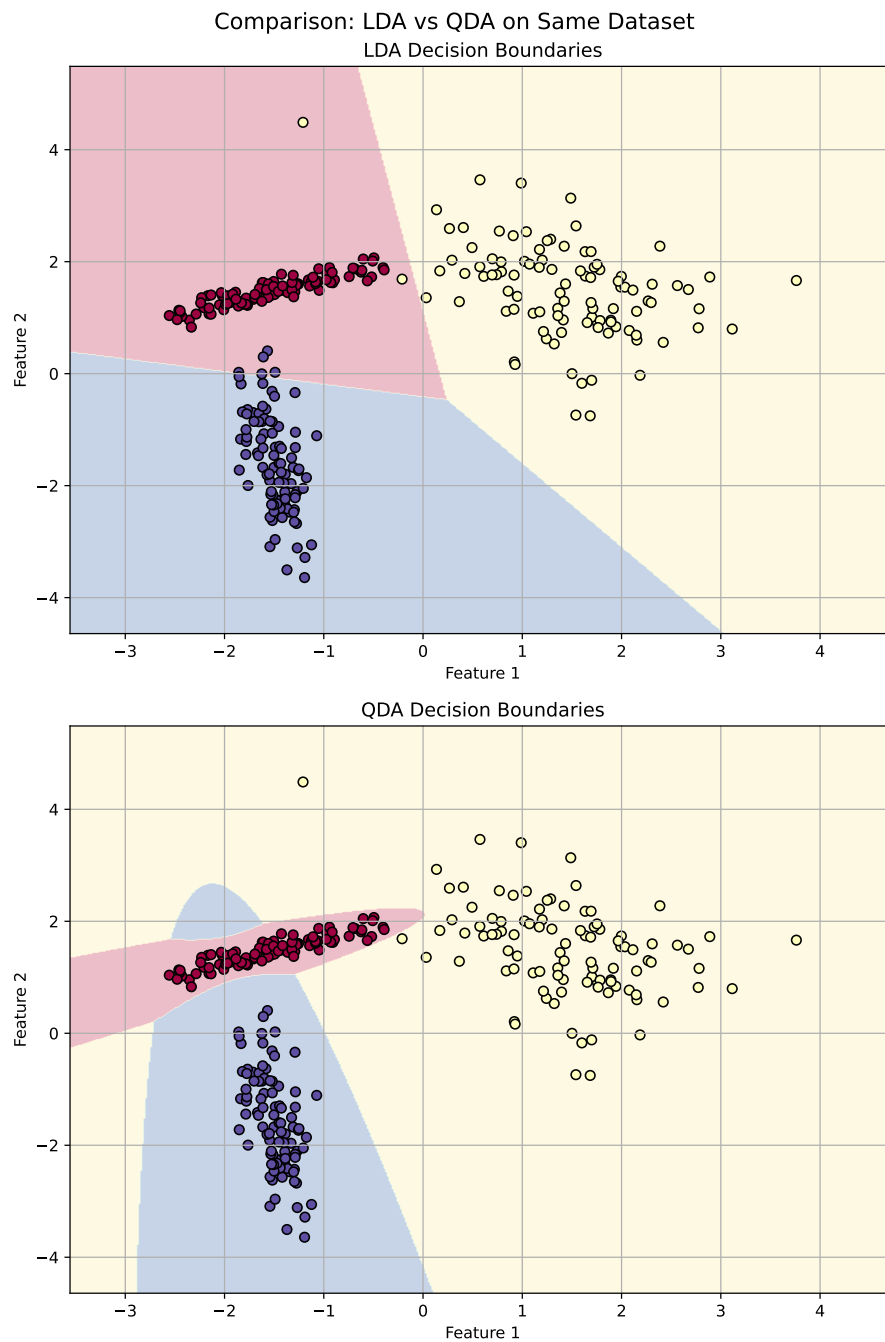
Figure 12: We have two clouds of points. In LDA, they are both ellipses of the same shape, just located in different places. In QDA, one ellipse might be wide and flat, the other tall and narrow, and the model adapts to those shapes.

### ❷ The Decision Rule and Why It's Quadratic

We want to classify a new point $\mathbf{x}$ by computing the **posterior probability** (same as page 76):

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right)$$

And assigning $\mathbf{x}$ to the class $k$ with the **highest posterior**. This is the same idea as LDA, but now, each class has **its own covariance matrix**, which changes the math.

1. **Bayes' Rule** (same as page 76).

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right) = \frac{\mathbb{P}\left(\mathbf{x} \mid y = k\right) \cdot \mathbb{P}\left(y = k\right)}{\mathbb{P}\left(\mathbf{x}\right)}$$

Again, since the denominator $\mathbb{P}\left(\mathbf{x}\right)$ is the same for all classes, we only need to **maximize the numerator**:

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right) \propto \pi_k \cdot \phi\left(\mathbf{x} \mid \mu_k, \Sigma_k\right)$$

- $\pi_k$ is the class prior.
- $\phi\left(\mathbf{x} \mid \mu_k, \Sigma_k\right)$ is the Gaussian density with **class-specific** $\Sigma_k$

2. **Log of the Gaussian Density**. Taking the **log** of this expression (just like in LDA, page 77), we get a **Discriminant Function**:

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \cdot \log\left|\Sigma_k\right| - \frac{1}{2} \cdot \left(\mathbf{x} - \mu_k\right)^T \Sigma_k^{-1}\left(\mathbf{x} - \mu_k\right) + \log\left(\pi_k\right) \qquad (58)$$

The **quadratic part** comes from:

$$\left(\mathbf{x} - \mu_k\right)^T \Sigma_k^{-1}\left(\mathbf{x} - \mu_k\right)$$

Which is a **quadratic form**. A **Quadratic Form** is just a fancy way to describe any expression where variables are multiplied together, especially with squares and cross-terms.

The **decision boundary** between class $k$ and $l$ is:

$$\delta_k\left(\mathbf{x}\right) = \delta_l\left(\mathbf{x}\right)$$

This becomes a **quadratic equation** in $\mathbf{x}$. Hence, the boundary is a **curve**, not a straight line (in 2D curved lines, in 3D curved surfaces, and in $p$-D quadratic hypersurfaces).

## ✅ When to Prefer QDA Over LDA

✔ **Class Covariances Are Clearly Different**. If our data shows that:

  – Class A is tightly clustered.
  – Class B is widely spread out.
  – Or they have very different **orientations**.

Then QDA is the better choice because it allows:

$$\Sigma_1 \neq \Sigma_2 \neq \ldots \neq \Sigma_K$$

This leads to **more accurate boundaries**, especially when classes have **non-linear separation**.

✔ **We Have Enough Data**. QDA estimates **a full covariance matrix for each class**, which means:

  – More parameters to estimate.
  – Higher risk of overfitting if sample size is small

So we **use QDA when our dataset is large enough** to support all those estimates reliably.

✔ **We Suspect Curved Boundaries**. If we visually or intuitively suspect:

  1. The data cannot be separated with a straight line.
  2. There's curvature in how the classes divide.

Then QDA is more likely to model the real structure.

## ⚠ When to Avoid QDA

✖ **Small sample size**. Estimating separate covariance matrices can become unstable.

✖ **High-dimensional data**. In $p$-dimensional space, each $\Sigma_k$ has $\dfrac{p\,(p+1)}{2}$ parameters, QDA becomes data-hungry.

✖ **Risk of overfitting**. More flexibility, then higher variance, especially with noise or unbalanced classes.

| Use **LDA** when... | Use **QDA** when... |
|---|---|
| We want simplicity and robustness | We want flexibility and accuracy |
| Covariances are similar | Covariances are different |
| Data is limited | We have enough samples per class |
| Boundaries are approximately linear | Boundaries appear curved |

## Example 4: Overfitting
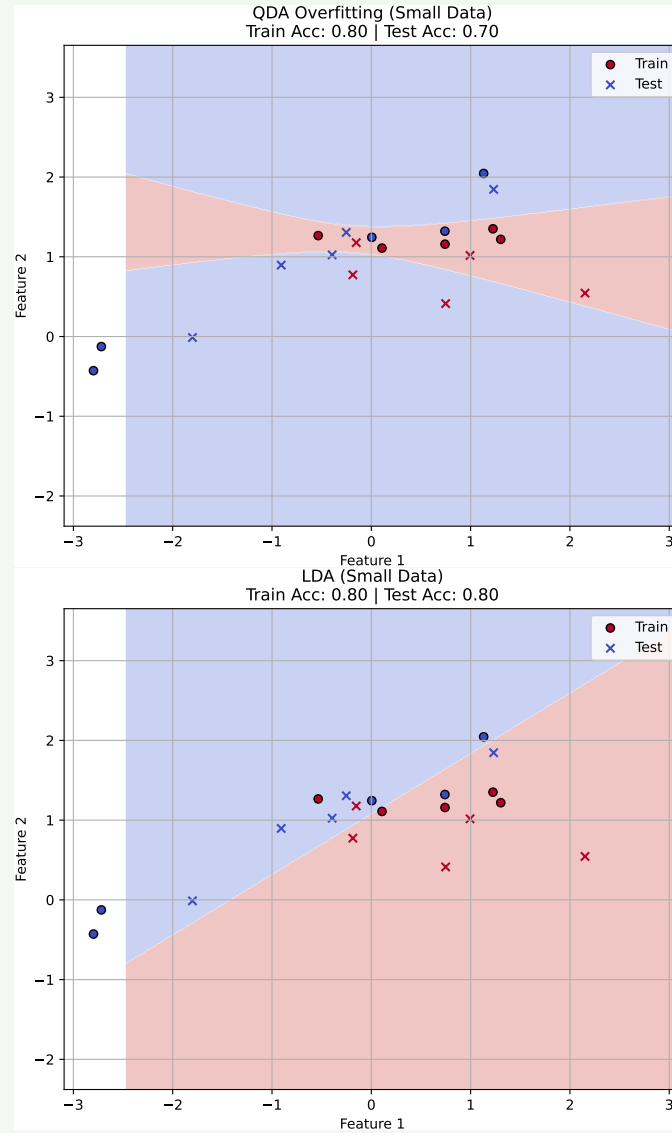


In this example, both QDA and LDA were trained on a very small dataset with only 10 training samples per class. QDA achieved 80% accuracy on the training set by fitting a flexible, curved decision boundary that tightly adapted to the few observed points. However, this resulted in poor generalization, with test accuracy dropping to just 70%, a possible case of overfitting. In contrast, LDA used a simpler, linear decision boundary, achieving same training accuracy (80%) but significantly better performance on the test set (80%). This demonstrates that LDA, being more constrained, generalizes better in low-data regimes, whereas QDA requires more data to avoid overfitting its more complex model.

## 3.5   Comparison LDA vs QDA

### ✖ Core Similarities

| Aspect | Description |
|---|---|
| **Both are generative** | They model how each class generates data using a multivariate Gaussian. |
| **Both use Bayes' rule** | Classification is based on computing posterior probabilities. |
| **Both are supervised** | They require labeled training data. |
| **Both work for multiple classes** | Not limited to binary classification. |

### ⚖ Key Differences

| Feature | **LDA** | **QDA** |
|---|---|---|
| **Covariance assumption** | Same for all classes: $\Sigma$ | Unique for each class: $\Sigma_k$ |
| **Decision boundary** | Linear (straight lines/planes) | Quadratic (curved boundaries) |
| **Model complexity** | Fewer parameters, then simpler | More parameters, then more flexible |
| **Training data needed** | Works well with small datasets | Needs more data (per class) |
| **Risk of overfitting** | Lower | Higher |
| **Bias-Variance tradeoff** | Higher bias, lower variance | Lower bias, higher variance |
| **Shape of decision region** | All classes have same shape | Each class can have different shape/size/orientation |

### ✔ Rule of Thumb

| Situation | Recommended |
|---|---|
| We want a **stable model** with limited data | Use **LDA** |
| We have **enough data** and suspect curved decision boundaries | Try **QDA** |
| Our classes look like **different clouds** (e.g. rotated, stretched) | Use **QDA** |
| Our classes are well-separated by a **straight line/plane** | **LDA** is likely sufficient |

## 3.6  Bayes Classifier

### 3.6.1  What is the Bayes Classifier?

The **Bayes Classifier** is a **decision rule** that assigns each observation to the class with the **highest posterior probability**, based on the conditional probability distribution of the features given the class.

√× **Formal Definition**

Let:

- $\mathbf{x} \in \mathbb{R}^p$ be a new observation (feature vector).

- $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$ be the possible **classes**.

- $\pi_k = P(Y = \mathcal{C}_k)$ be the **prior probability** of class $k$.

- $f_k(\mathbf{x}) = P(\mathbf{X} = \mathbf{x} \mid Y = \mathcal{C}_k)$ be the **class-conditional density**.

Then the **posterior probability** of class $\mathcal{C}_k$ given $\mathbf{x}$ is:

$$P(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}) = \frac{\pi_k f_k(\mathbf{x})}{\displaystyle\sum_{j=1}^{K} \pi_j f_j(\mathbf{x})} \tag{59}$$

The **Bayes classifier** assigns $\mathbf{x}$ to the class with the **highest posterior**:

$$\hat{y} = \arg\max_k P(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}) \tag{60}$$

✅ **Math Behind the Bayes Classifier**

We will try to explain it in the simplest terms possible. Imagine we're a doctor trying to decide if a patient has **flu** or **allergy**, based on their **temperature**. We know from past experience:

- People with the flu usually have higher temperatures.

- People with allergies usually have normal temperatures.

Now, a new patient walks in with a temperature of 37.8°C. We ask: "*given this temperature, what is the most likely diagnosis?*". The Bayes Classifier answer this by saying "*pick the class (flu or allergy) that is most likely, based on what we know about temperatures in each class.*". So:

- If flu patients often have temperatures around 37.8°C, and allergy patients don't;

- Then the Bayes Classifier says: "*assign this patient to the flu class.*".

Now that the intuition is clear, let's explain the math behind the Bayes Classifier.

We are trying to **classify a new observation x**, such as someone's temperature, into one of $K$ classes (e.g., flu vs allergy). We want to choose the class $\mathcal{C}_k$ that **maximizes the probability that it's the correct class**, given the data. This is the **posterior probability** (page 76):

$$P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right)$$

To compute this, we use **Bayes' theorem**:

$$P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right) = \frac{\pi_k f_k\left(\mathbf{x}\right)}{\displaystyle\sum_{j=1}^{K} \pi_j f_j\left(\mathbf{x}\right)} \tag{61}$$

Let's explain each term:

- $\pi_k = P(Y = \mathcal{C}_k)$ is the **prior**, *how common is class $k$ in general?* For example, if 40% of patients have the flu, then $\pi_{\text{flu}} = 0.4$.

- $f_k(\mathbf{x}) = P(\mathbf{X} = \mathbf{x} \mid Y = \mathcal{C}_k)$ is the **Class-conditional density**. For example, how likely is temperature 37.8°C *if* someone has the flu?

- The **numerator** $\pi_k f_k(\mathbf{x})$ tells us: how likely is it that someone is in class $k$ and has this data $\mathbf{x}$?

- The **denominator** just makes sure that all the probabilities sum to 1.

Although the formula may be complex, we **don't need to compute the entire formula**. Since the denominator is the same for all classes, we only care about comparing:

$$\pi_k f_k\left(\mathbf{x}\right)$$

We choose the class $k$ that **maximizes** this value:

$$\hat{y} = \arg\max_k \ \pi_k f_k\left(\mathbf{x}\right)$$
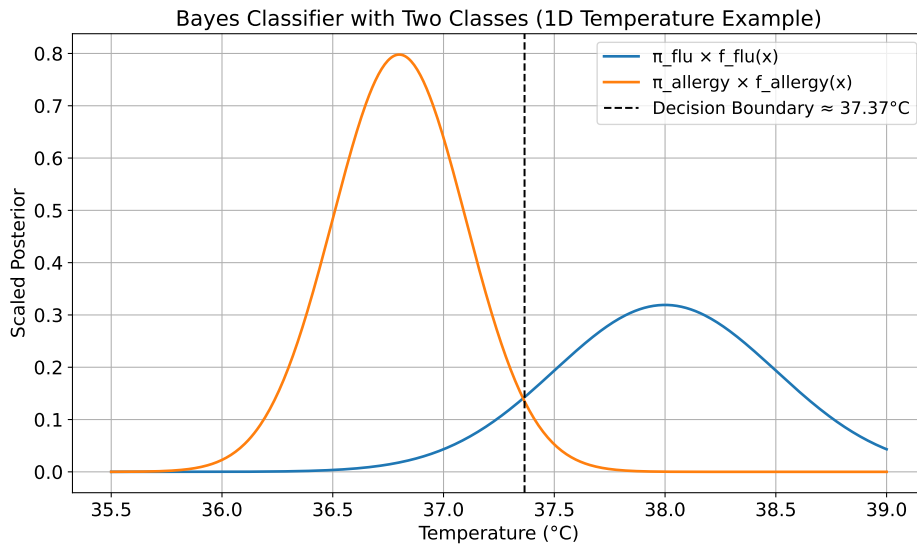
Figure 13: This plot shows how the Bayes classifier works in a simple case with two classes (flu and allergy), and one feature (body temperature). The blue curve $\pi_{\text{flu}} \cdot f_{\text{flu}}(x)$ is the likelihood of flu at each temperature, and the orange curve $\pi_{\text{allergy}} \cdot f_{\text{allergy}}(x)$ is the likelihood of allergy. At any temperature $x$, we compare the two curves; whichever is higher, we choose that class. The black dashed line shows the Bayes decision boundary, around 37.37°C: for $x < 37.37$ choose allergy, otherwise for $x > 37.37$ choose flu.

# References

[1] Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American statistical Association*, 68(342):361–368, 1973.

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer Series in Statistics. Springer New York, 2009.

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in Python.* Springer Texts in Statistics. Springer New York, 2013.

[4] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis.* Applied Multivariate Statistical Analysis. Pearson Prentice Hall, 2007.

[5] Beraha Mario. Applied statistics. Slides from the HPC-E master's degree course on Politecnico di Milano, 2024-2025.

# Index