

# Week - 1: Unsupervised Learning

## What is Clustering?

**Clustering** is an **unsupervised learning** technique used to find **patterns or groups** (called **clusters**) in a dataset **without labels**.

---

## Supervised vs. Unsupervised Learning

| Aspect                | Supervised Learning                            | Unsupervised Learning          |
|-----------------------|------------------------------------------------|--------------------------------|
| Data includes labels? | ✓ Yes (features <b>x</b> and labels <b>y</b> ) | ✗ No (only features <b>x</b> ) |
| Goal                  | Learn to predict labels                        | Find patterns/structure        |
| Example               | Binary classification                          | Clustering                     |

---

## What Does Clustering Do?





- Groups similar data points together.
  - Identifies **structure** in the data.
  - Finds **natural groupings** without knowing any labels.
- 

## Example:

Imagine a dataset of dots plotted in 2D space.

- In **supervised learning**, you know which dot is which class (e.g., red vs. blue).
  - In **clustering**, you don't know that — you ask the algorithm to group the dots based on similarity.
-

## Applications of Clustering:

-  **News article grouping** (e.g., similar topics like science or sports)
-  **Market segmentation** (e.g., learners with different goals)
-  **Genetic data analysis** (e.g., finding people with similar traits)
-  **Astronomy** (e.g., grouping stars or galaxies)

### ◆ K-means Clustering

**K-means** is an unsupervised algorithm that groups data into clusters based on similarity.

#### ◆ K-means Algorithm Summary

1. **Initialize Centroids:**  
Randomly choose  $K$  cluster centers (centroids).
2. **Repeat Until Convergence:**
  - **Assign Points:** Each data point is assigned to the nearest centroid.
  - **Update Centroids:** Move each centroid to the average of the points assigned to it.
3. **Special Case:**  
If a centroid has no points, either remove it or reinitialize it randomly.
4. **Convergence:**  
The algorithm stops when point assignments and centroid positions no longer change.
5. **Use Case:**  
Works well even without clear clusters—like grouping people into t-shirt sizes using height and weight.

## K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K$

Repeat {

*# Assign points to cluster centroids*

for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid closest to  $x^{(i)}$

*# Move cluster centroids*

for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

}

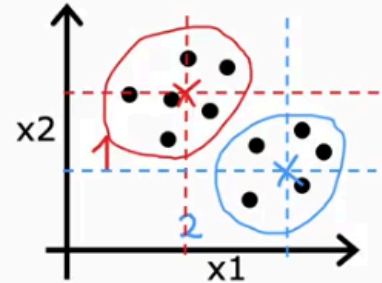
$$\mu_1 = \frac{1}{4} [x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}]$$

$\mu_1, \mu_2$

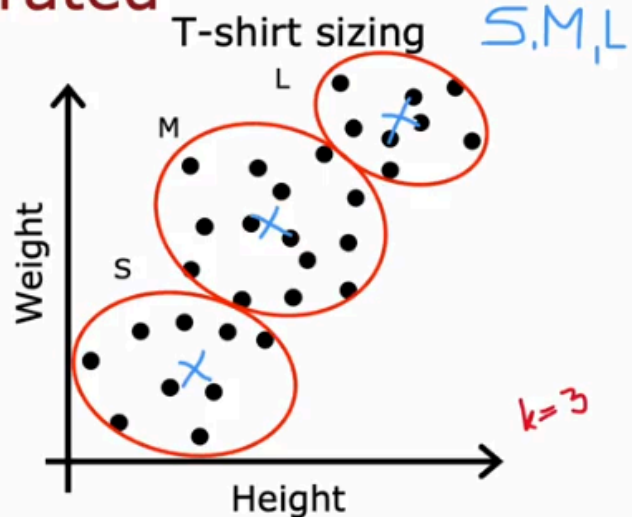
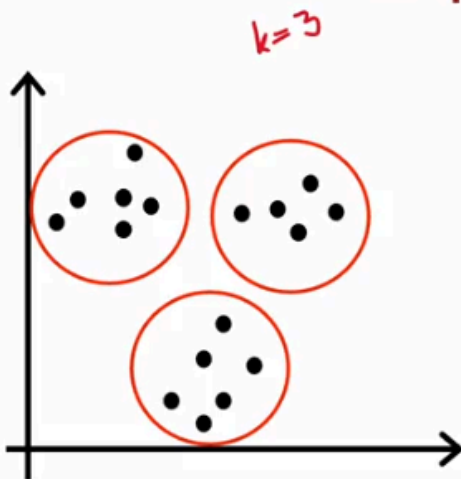
$x^{(1)}, x^{(2)}, \dots, x^{(n)}$

$n=2$

$K=2$



## K-means for clusters that are not well separated



### ✓ Final Result:

- Points are grouped into  $k$  clusters.
- The algorithm stops when the clusters stabilize (no more changes).

## K-means optimization objective

$c^{(i)}$  = index of cluster  $(1, 2, \dots, K)$  to which example  $x^{(i)}$  is currently assigned

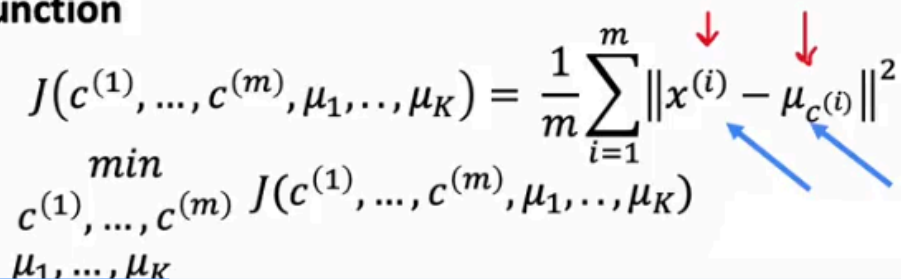
$\mu_k$  = cluster centroid  $k$

$\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned

**Cost function**

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$



### ◆ K-means is Also an Optimization Algorithm

In supervised learning (like linear regression), you define a **cost function** and then **optimize** it (e.g., with gradient descent). Similarly, **K-means** also **minimizes a cost function**, but it uses a different method than gradient descent.

### ◆ The Cost Function of K-means (Also Called the "Distortion Function")

The **cost function** ( $J$ ) in K-means is:

$$J = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Where:

- $x^{(i)}$ : the  $i$ -th data point
- $c^{(i)}$ : the index of the cluster assigned to  $x^{(i)}$
- $\mu_{c^{(i)}}$ : the centroid of the cluster that  $x^{(i)}$  is assigned to

👉 This is the **average squared distance** between each point and its assigned cluster centroid.

## ♦ Why the Two K-means Steps Minimize This Cost

### 1. Assign Points to Closest Centroids (Step 1)

For each point  $x(i)$  we assign it to the **closest** centroid.

✓ This **minimizes** the squared distance from that point to a centroid.

### 2. Move Centroids to the Mean of Assigned Points (Step 2)

After assigning points, update each centroid to be the **average of the points** assigned to it.

✓ This choice **minimizes** the average squared distance for that cluster.

---

## ♦ Why K-means Converges

- Each step of K-means **reduces or keeps the cost function the same**
  - Since the cost function can't go below 0, eventually the algorithm **must stop**
  - Once the cost stops decreasing → **K-means has converged**
- 

## ♦ Practical Use of Cost Function

- 🧠 **Use it to detect convergence:** When the cost stops changing, stop the algorithm.
  - 🎲 **Try multiple random initializations:** Run K-means several times with different random centroids and pick the run with the **lowest final cost** for better results.
- 

## ♦ Final Takeaway

K-means works by:

1. Assigning each point to its **closest centroid**
2. Moving centroids to the **average of assigned points**
3. Repeating until the **cost function (distortion) stops decreasing**

This process **optimizes** the clustering and guarantees convergence (though not always to the global best).

## ♦ K-means Initialization (Short Summary)

- K-means starts by **randomly selecting K training examples** as initial centroids.
- **Different random starts** can lead to **different clustering results** — some good, some bad.
- To get better results:
  - **Run K-means multiple times** (e.g., 100 times) with different initializations.
  - **Pick the best run** — the one with the **lowest cost (distortion J)**.
- This reduces the chance of getting stuck in a **bad local minimum** and gives **better clusters**.

## ♦ Practical Tip

- Doing multiple random initializations helps **avoid bad local minima** and usually gives **much better clusters**.
- 100 initializations is common; going beyond 1000 may give little improvement and take more time.

## ♦ How to Choose the Number of Clusters in K-means

The **K-means algorithm** requires you to **choose K (the number of clusters)** beforehand. But how do you know what the "right" **K** is?

### ✓ 1. No One Correct Answer

- In many real-life datasets, there's no clear or “correct” number of clusters.
- For the same data, some people might see **2 clusters**, others might see **4**, and both can be valid.
- This is because **clustering is unsupervised** — there are no labels to compare with.

### ✓ 2. Elbow Method

- One method used in research is the **elbow method**:
  - Run K-means with various values of **K** (e.g., 1 to 10).

- Plot the **cost function (J)** against **K**. This function shows the average squared distance between points and their assigned centroids.
  - Initially, as **K** increases, the cost reduces sharply.
  - At some point, the cost starts decreasing more slowly — this point is called the “**elbow**”, and it's a suggested value for **K**.
- ♦ But this method doesn't always work. In practice, many plots **don't show a clear elbow**, so it can still be ambiguous.

### ✗ 3. What Not to Do

- Don't pick **K** just by **minimizing the cost function**.
- More clusters **always reduce the cost**, but it can **overfit** or **make things unnecessarily complex**.

### ✓ 4. Better Strategy: Choose **K** Based on Your Goal

- Think about **how you'll use the clusters**:
    - Example: **T-shirt sizing**
      - $K=3 \rightarrow$  Small, Medium, Large
      - $K=5 \rightarrow$  XS, S, M, L, XL
    - Both options are valid. But more clusters mean **better fit**, while fewer clusters mean **lower cost and easier production**.
    - So you pick the **K** that makes the most **business or practical sense**.
  - Another example: **image compression**
    - More clusters = better image quality, but larger file size.
    - Fewer clusters = smaller file, but lower image quality.
    - Choose **K** based on your **trade-off between quality and size**.
-

## Final Thoughts

- There is **no universal rule** to pick the perfect  $K$ .
- It depends on the **use case**, **trade-offs**, and **what you want to do** with the clusters.
- Try different values and choose what works **best for your specific goal**.

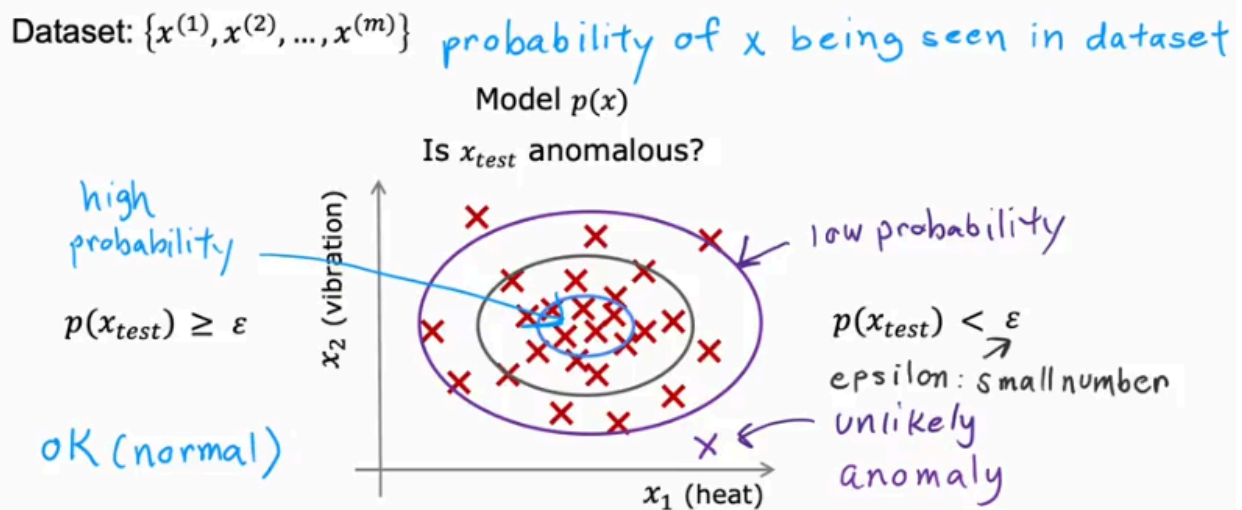
## What is Anomaly Detection?

Anomaly detection is a technique in **unsupervised learning** where the algorithm learns from a dataset of **normal (non-anomalous)** events to identify **unusual or abnormal instances** that deviate significantly from what's considered normal.

### How Anomaly Detection Works (Intuition)

1. The algorithm uses the **training data** to model the **probability distribution  $p(x)$** .
2. This distribution tells us  
What values of features ( $x_1, x_2$ ) are **common** (high probability) vs **rare** (low probability)?
3. If a new engine's feature vector  $X_{\text{test}}$  has a **low probability**, we flag it as **anomalous**.

## Density estimation



## Threshold for Anomaly

- We define a small number  $\epsilon$  (**epsilon**) as a threshold.



- If:  
 $p(X_{\text{test}}) < \epsilon \rightarrow$  it's **an anomaly**  
 $p(X_{\text{test}}) \geq \epsilon \rightarrow$  it's **probably normal**

## Visualization

- Normal data clusters together (e.g., in ellipses in the center of the plot).
- Anomalies lie **far outside**, where probability  $p(x)$  is very low.

## Applications of Anomaly Detection

Anomaly detection is used in **many real-world domains**, such as:

1. **Fraud Detection:**
  - Features: login frequency, typing speed, number of transactions.
  - Goal: Spot unusual behavior to flag potentially fraudulent activity.
  - Action: Flag for human review or extra verification steps.
2. **Manufacturing:**
  - Examples: engines, smartphones, motors, PCBs.
  - Goal: Detecting faulty units **before** they are shipped.
3. **Computer/Data Center Monitoring:**
  - Features: CPU load, memory usage, disk activity.
  - Goal: Detect failing or compromised machines.
4. **Telecom (Telco) Networks:**
  - Used to detect faulty **cell towers** so that technicians can fix them.

## Anomaly detection example

Fraud detection:

- $x^{(i)}$  = features of user  $i$ 's activities
- Model  $p(x)$  from data.
- Identify unusual users by checking which have  $p(x) < \epsilon$

*perform additional checks to identify real fraud vs. false alarms*

Manufacturing:

$x^{(i)}$  = features of product  $i$

*airplane engine  
circuit board  
smartphone*

*ratios*

Monitoring computers in a data center:

$x^{(i)}$  = features of machine  $i$

- $x_1$  = memory use,
- $x_2$  = number of disk accesses/sec,
- $x_3$  = CPU load,
- $x_4$  = CPU load/network traffic.

*how often log in?  
how many web pages visited?  
transactions?  
posts? typing speed?*

## 🔍 What is the Gaussian/Normal Distribution?

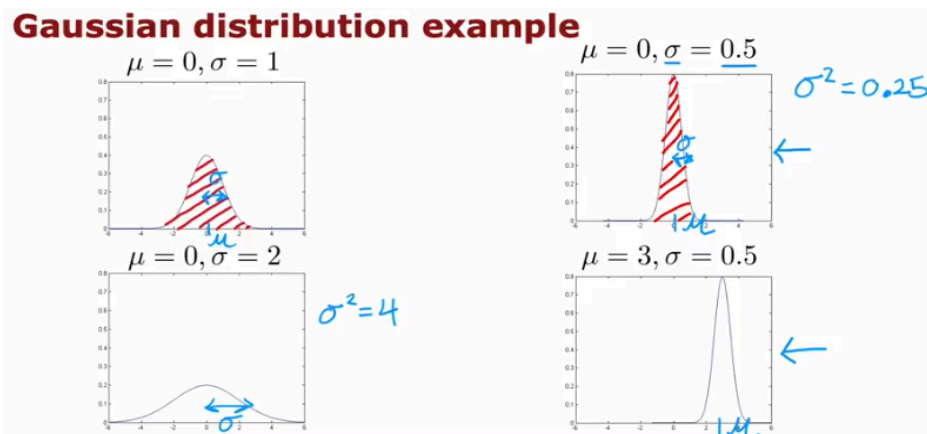
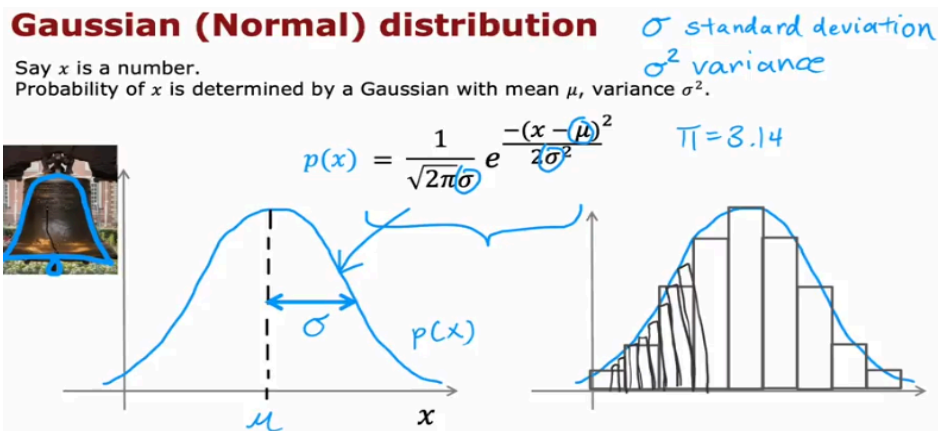
- It's a **probability distribution** used to describe data that clusters around a **mean (average)** value.
- Also known as the **bell-shaped curve**, because of its shape.
- It's commonly used in statistics, machine learning, and anomaly detection.

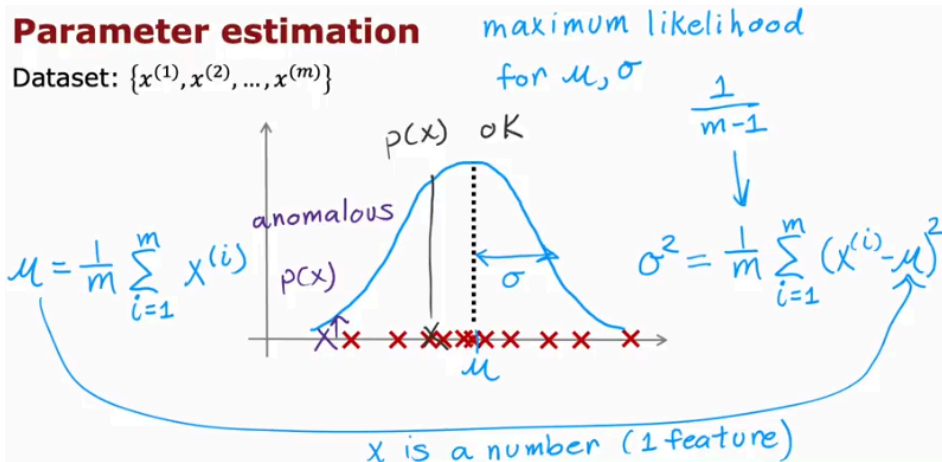
### 📊 Key Parameters of a Gaussian Distribution:

- $\mu$  (**Mu**): the **mean** – the center of the distribution.
- $\sigma$  (**Sigma**): the **standard deviation** – measures how spread out the data is.
- $\sigma^2$  (**Sigma squared**): the **variance** – it's just the square of the standard deviation./**average squared difference from the mean.**

So:

- If  $\sigma$  is **small**, the curve is **narrow and tall**.
- If  $\sigma$  is **large**, the curve is **wide and short**.
- If  $\mu$  changes, the whole curve **shifts left or right**.





## Anomaly Detection Multiple features:

### Steps to Build the Algorithm

1. Training Data  
Each example  $x^{(i)}$  has  $n$  features  $\rightarrow x = [x_1, x_2, \dots, x_n]$
2. Assume Independence  
Model the overall probability as:  $p(x) = p(x_1) \cdot p(x_2) \cdot \dots \cdot p(x_n)$
3. Model Each Feature

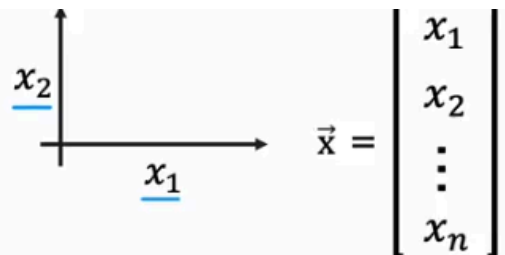
Assume each feature follows a Gaussian (Normal) distribution:

$$p(x_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \cdot e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}}$$

4. Estimate Parameters  
From training data, compute:
  - a.  $\mu_j$ : mean of feature  $j$
  - b.  $\sigma_j^2$ : variance of feature  $j$
5. Evaluate New Examples  
For a test point  $x$ , compute  $p(x)$  as the product of all  $p(x_j)$
6. Anomaly Decision  
Set a threshold  $\epsilon$ :
  - a. If  $p(x) < \epsilon \rightarrow$  Anomaly
  - b. Else  $\rightarrow$  Normal

## Density estimation

Training set:  $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$   
 Each example  $\vec{x}^{(i)}$  has  $n$  features



$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * p(x_3; \mu_3, \sigma_3^2) * \dots * p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

$\sum$  "add"       $\prod$  "multiply"

$$p(x_1 = \text{high temp}) = 1/10$$

$$p(x_2 = \text{high vibra}) = 1/20$$

$$p(x_1, x_2) = p(x_1) * p(x_2)$$

$$= \frac{1}{10} * \frac{1}{20} = \frac{1}{200}$$

## Anomaly detection algorithm

1. Choose  $n$  features  $x_i$  that you think might be indicative of anomalous examples.

2. Fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

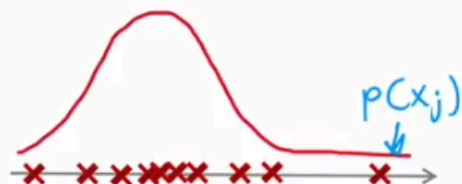
Vectorized formula

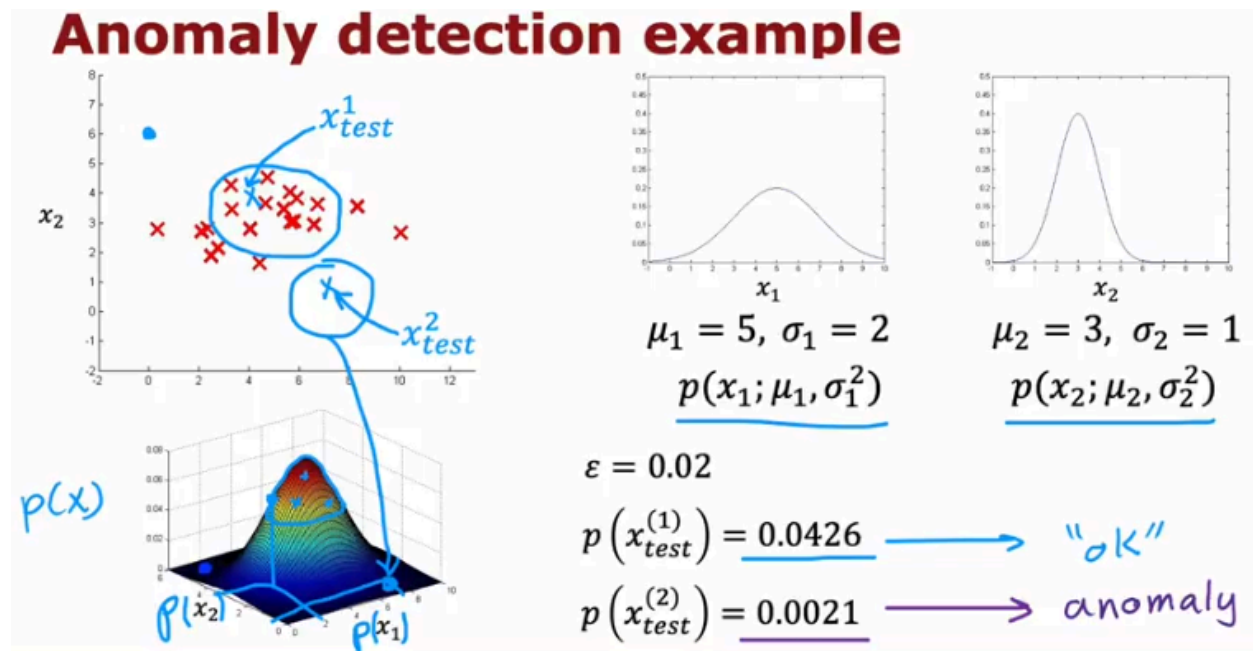
$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_n \end{bmatrix}$$

3. Given new example  $x$ , compute  $p(x)$ :

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if  $p(x) < \epsilon$





**This passage offers practical tips for developing and evaluating an anomaly detection system:**

1. **Real-Number Evaluation:** During development, evaluating the algorithm with real-number metrics helps make decisions about parameter tuning (e.g., adjusting **epsilon**) by comparing how well it detects anomalies.
2. **Cross-Validation and Test Sets:** Use a **cross-validation set** and **test set** containing both normal data and known anomalies to tune and evaluate the algorithm's performance. Cross-validation helps adjust parameters, and the test set checks generalization to unseen data.
3. **Alternative with Few Anomalies:** If anomalies are few, you might combine the cross-validation and test sets, but this could lead to overfitting and reduced real-world performance.
4. **Evaluation Metrics:** Use metrics like **precision**, **recall**, and **F1 score** instead of accuracy when the data is highly skewed (more normal examples than anomalies) to assess how well the algorithm detects anomalies.
5. **Labeled Data:** Having a small number of labeled anomalies helps tune the system more effectively. The algorithm is still unsupervised, with labeled data used only for evaluation and tuning.
6. **Anomaly Detection vs. Supervised Learning:** The passage hints that the use of labeled data might lead to considering **supervised learning** instead of anomaly detection, which will be discussed in the next video.

## Anomaly detection vs. supervised learning

### Key Points:

#### 1. **Anomaly Detection:**

- Best for situations where there are **very few positive examples** (0-20), and **many negative examples**.
- It assumes **new types of anomalies** may appear that were never seen before. The algorithm learns from normal data (negative examples) and flags deviations as anomalies, even if the anomaly is entirely new.

#### 2. **Supervised Learning:**

- More appropriate when there are **more positive examples**.
- Assumes **future positive examples** will resemble those in the training set. Works best if you have enough labeled data to predict outcomes based on similarities to previous examples.

#### 3. **When to Choose Each:**

- **Anomaly Detection:** Works well when dealing with unknown, new types of anomalies. Examples include fraud detection (e.g., financial fraud) where new fraud attempts keep emerging, and manufacturing defects where new types of failures may appear.
- **Supervised Learning:** Works best for problems with **repeated, known patterns**. Examples include email spam detection (where spam is often similar over time) and predicting the weather (where patterns repeat).

#### 4. **Manufacturing Example:**

- If the defect (e.g., scratched smartphones) is well-known, supervised learning is ideal, as enough labeled data exists.
- If the defect is unknown and could be something new, anomaly detection is more appropriate.

#### 5. **Security Applications:**

- In cybersecurity, new types of attacks emerge frequently, so anomaly detection is often used to identify abnormal patterns in machine behavior or potential hacks.

#### 6. **Choosing Between the Two:**

- **Anomaly Detection:** Finds novel anomalies that haven't been encountered before.
- **Supervised Learning:** Predicts based on patterns from previous data.

| Anomaly detection                                                                                       | vs. | Supervised learning                                                                       |
|---------------------------------------------------------------------------------------------------------|-----|-------------------------------------------------------------------------------------------|
| → Fraud detection                                                                                       |     | → Email spam classification                                                               |
| → Manufacturing - Finding <u>new previously unseen defects</u> in manufacturing.(e.g. aircraft engines) |     | → Manufacturing - Finding known, previously <u>seen</u> defects $y=1$<br><u>scratches</u> |
| → Monitoring machines in a data center                                                                  |     | → Weather prediction (sunny/rainy/etc.)                                                   |
|                                                                                                         |     | → Diseases classification                                                                 |

## Choosing what features to use

### 1. Feature Selection Matters More

- In anomaly detection (**unsupervised**), choosing the right features is critical since there are no labels to guide the model.

### 2. Make Features Gaussian

- **Gaussian distribution** is ideal for anomaly detection algorithms.
- Algorithms assume data is Gaussian. Use transformations to make feature distributions more bell-shaped:
  - **Log transformation** (e.g.,  $\log(X)$  or  $\log(X + c)$ )
  - **Square root transformation** (e.g.,  $X^{1/2}$ )
  - **Power transformation** (e.g.,  $X^{0.4}$ )

### 3. Plot & Transform

- Use histograms to visualize feature distributions.
- Try different transformations until the distribution looks Gaussian.

### 4. Error Analysis

- If the model fails on some anomalies, analyze those examples.
- Add or create new features that highlight what makes them different.

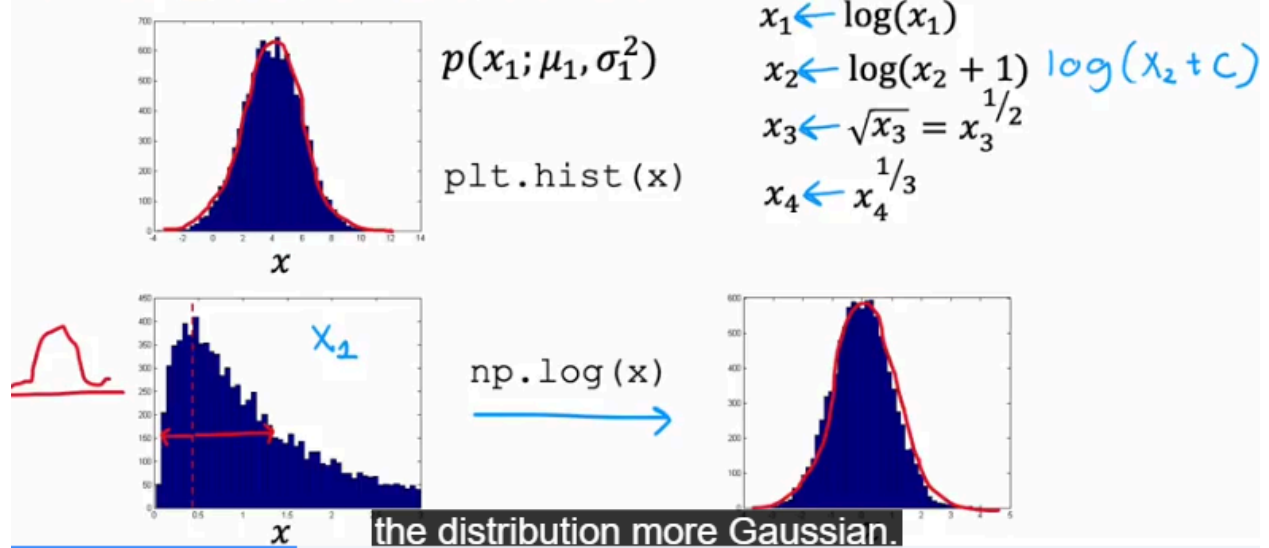
### 5. Create New Features

- Combine existing features (e.g., **CPU / Network Traffic**) to better capture anomalies.

### 6. Apply Changes Consistently

- Use the same transformations on training, validation, and test sets.

## Non-gaussian features



## Week - 2: Unsupervised Learning (Recomenders)