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 **x** and labels **y**)	❌ No (only features **x**)
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:

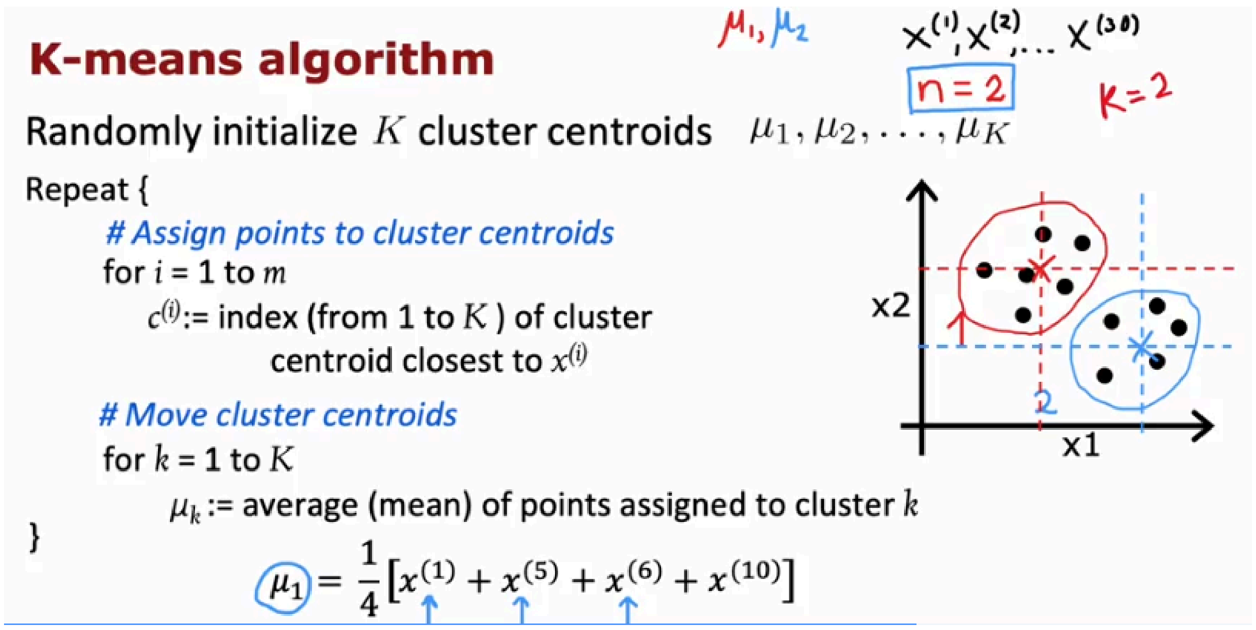
- 📰 **News article grouping** (e.g., similar topics like science or sports)
- 📈 **Market segmentation** (e.g., learners with different goals)
- 🧬 **Genetic data analysis** (e.g., finding people with similar traits)
- 🌌 **Astronomy** (e.g., grouping stars or galaxies)

🔹 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.

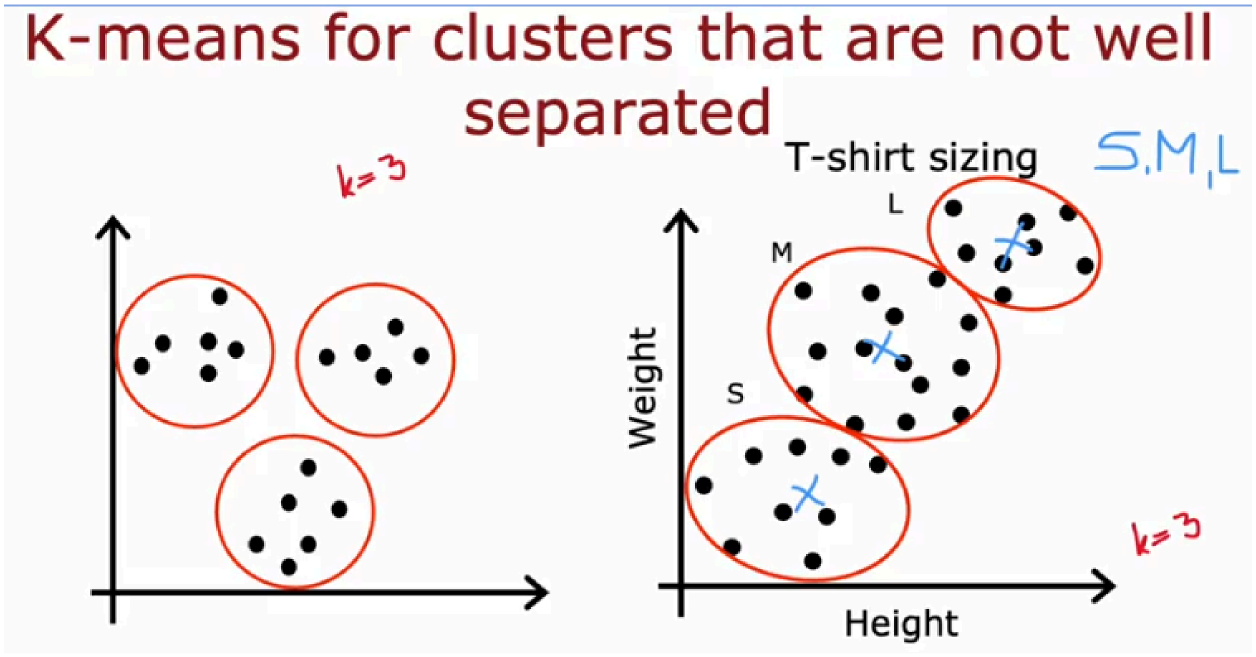


K-means algorithm

μ_1, μ_2 $x^{(1)}, x^{(2)}, \dots x^{(30)}$ $\boxed{n=2}$ $K=2$

```
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 $k$
}
```

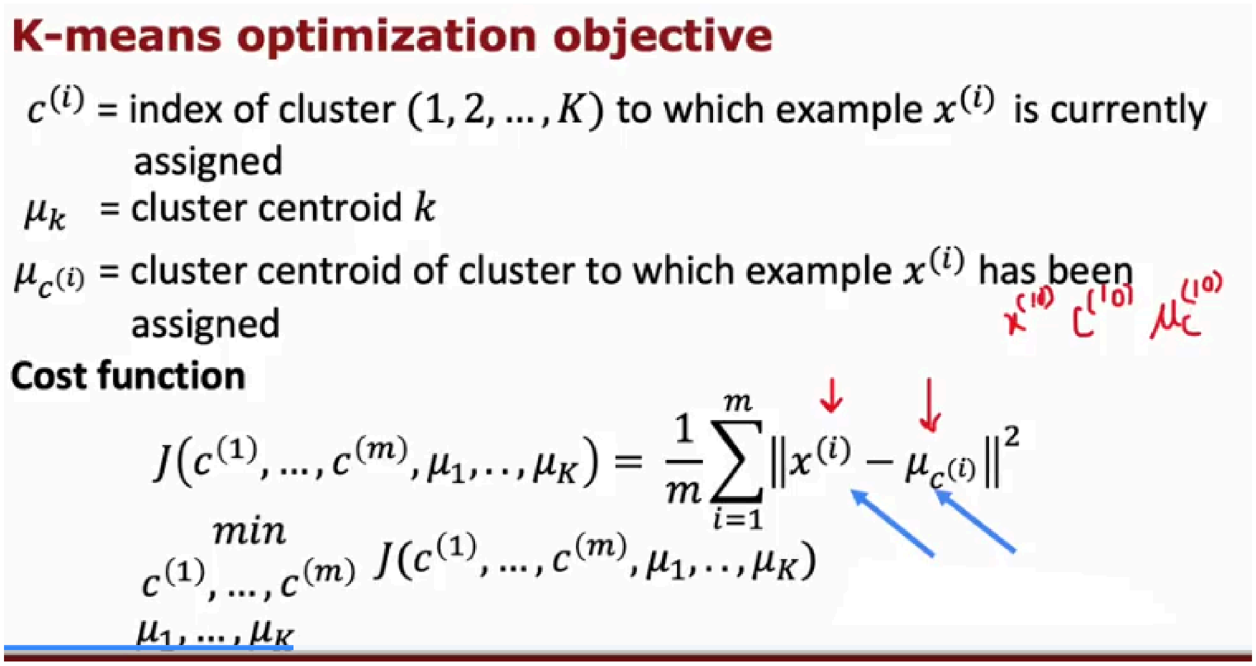
$$\mu_1 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right]$$



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 $(1, 2, \ldots, K)$ to which example $x^{(i)}$ is currently assigned

μ_k = cluster centroid k

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

$x^{(10)}$ $c^{(10)}$ $\mu_{c^{(10)}}$

Cost function

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

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

🔹 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 → Small, Medium, Large
        - K=5 → 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

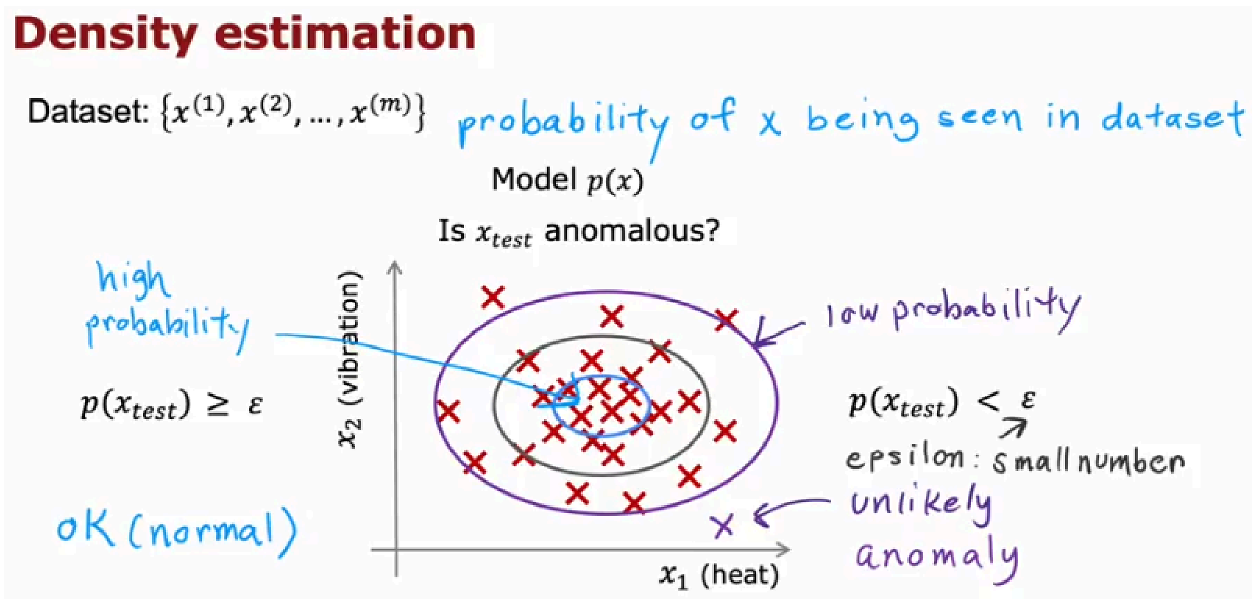
- There is **no universal rule** to pick the perfect K.
- It depends on the **use case**, **trade-offs**, and **what you want to do** with the clusters.
- Try different values and choose what works **best for your specific goal**.

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 (x1, x2) are **common** (high probability) vs **rare** (low probability)?
3. If a new engine's feature vector X_test has a **low probability**, we flag it as **anomalous**.



🎯 Threshold for Anomaly

- We define a small number **ε (epsilon)** as a threshold.
- If:
  p(X_test) < ε → it's **an anomaly**
  p(X_test) ≥ ε → it's **probably normal**

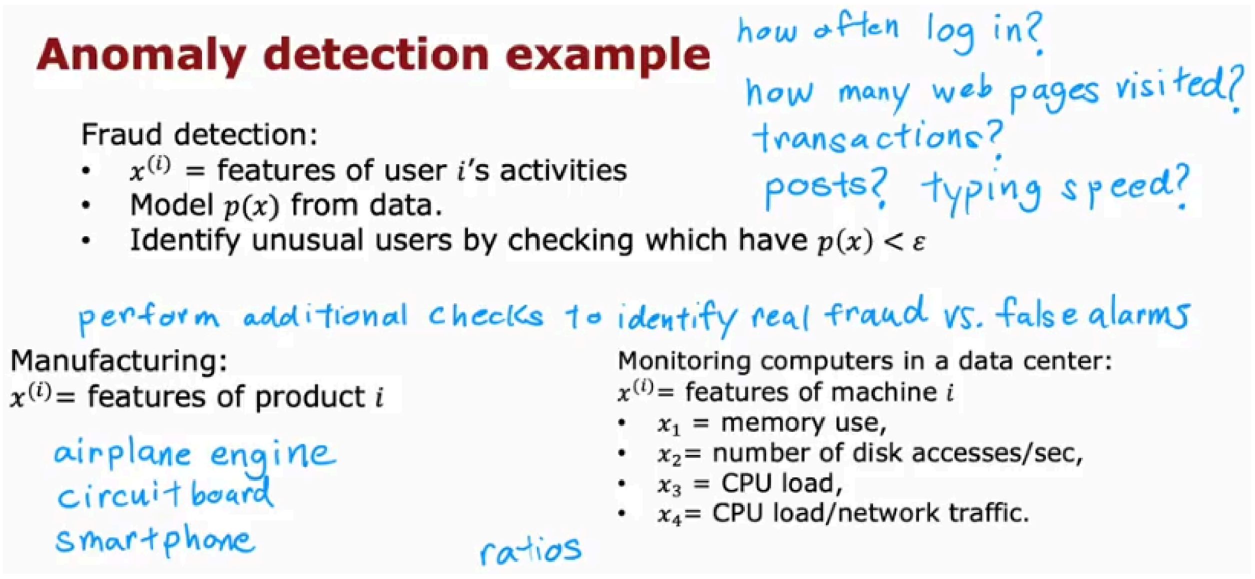
📈 Visualization

- 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

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

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

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.

🔍What is the Gaussian/Normal Distribution?

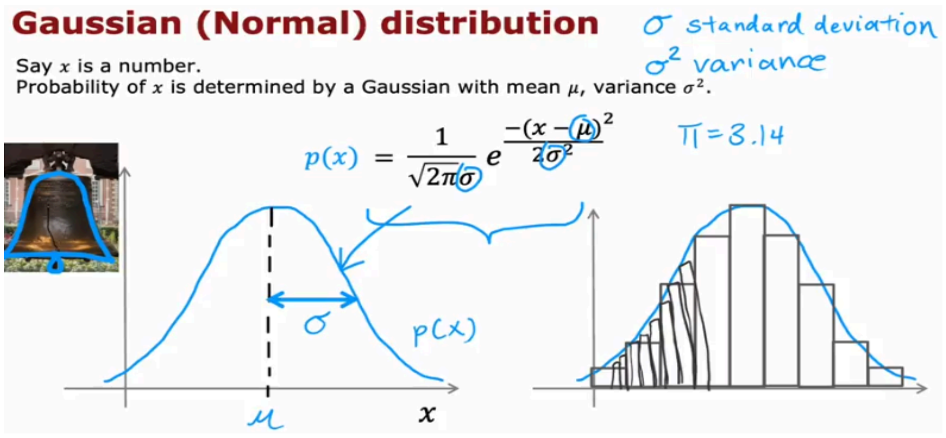
- It's a **probability distribution** used to describe data that clusters around a **mean (average)** value.
- Also known as the **bell-shaped curve**, because of its shape.
- It's commonly used in statistics, machine learning, and anomaly detection.

📊 Key Parameters of a Gaussian Distribution:

- **μ (Mu)**: the **mean** – the center of the distribution.
- **σ (Sigma)**: the **standard deviation** – measures how spread out the data is.
- **σ² (Sigma squared)**: the **variance** – it's just the square of the standard deviation./**average squared difference from the mean.**

So:

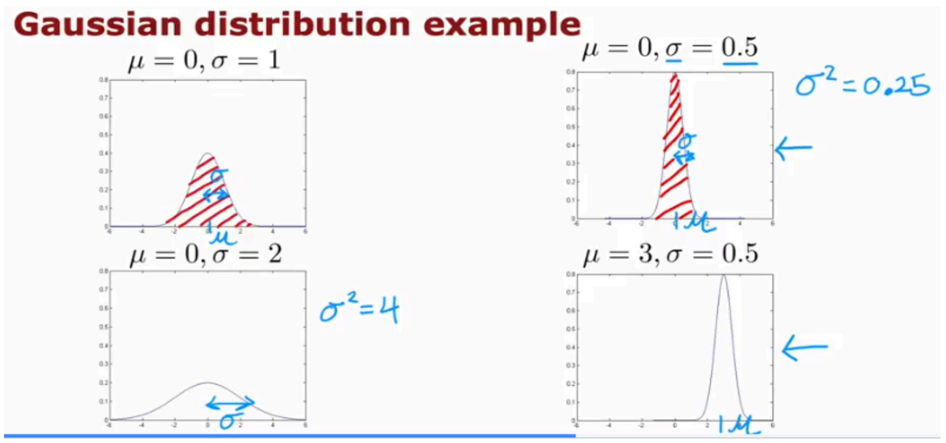
- If **σ is small**, the curve is **narrow and tall**.
- If **σ is large**, the curve is **wide and short**.
- If **μ changes**, the whole curve **shifts left or right.**



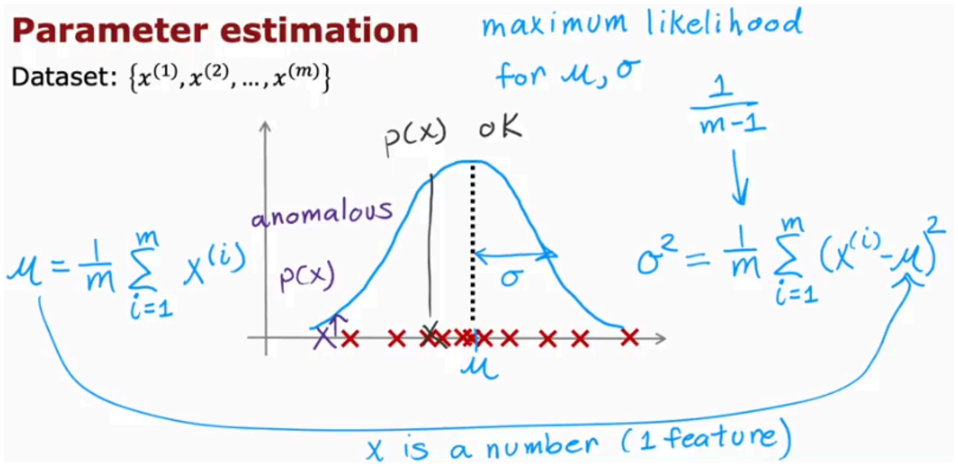
Gaussian (Normal) distribution

Say x is a number.
Probability of x is determined by a Gaussian with mean μ, variance σ^2.

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



Gaussian distribution example


Parameter estimation
Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
maximum likelihood for μ, σ
$\frac{1}{m-1}$
$p(x)$ ok
anomalous
$p(x)$
$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)^2$
σ
μ
x is a number (1 feature)

Anomaly Detection Multiple features:

🔧 Steps to Build the Algorithm

1. Training Data
   Each example x(i) has n features → x=[x1,x2,...,xn]
2. Assume Independence
   Model the overall probability as: p(x) = p(x1) · p(x2) · … · p(xn)
3. Model Each Feature

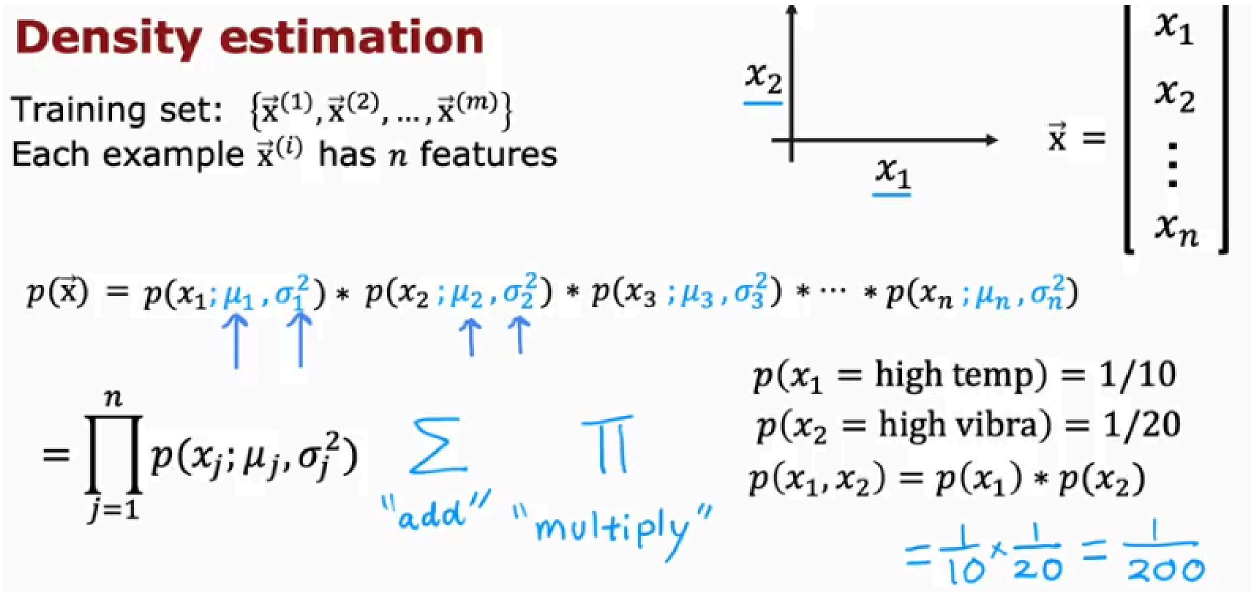
   Assume each feature follows a Gaussian (Normal) distribution:

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

4. Estimate Parameters
   From training data, compute:
   a. μj: mean of feature j
   b. σj2: variance of feature j

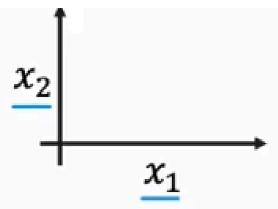
5. Evaluate New Examples
   For a test point x, compute p(x) as the product of all p(xj)

6. Anomaly Decision
   Set a threshold ε:
   a. If p(x) < ε → Anomaly
   b. Else → Normal



Density estimation

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



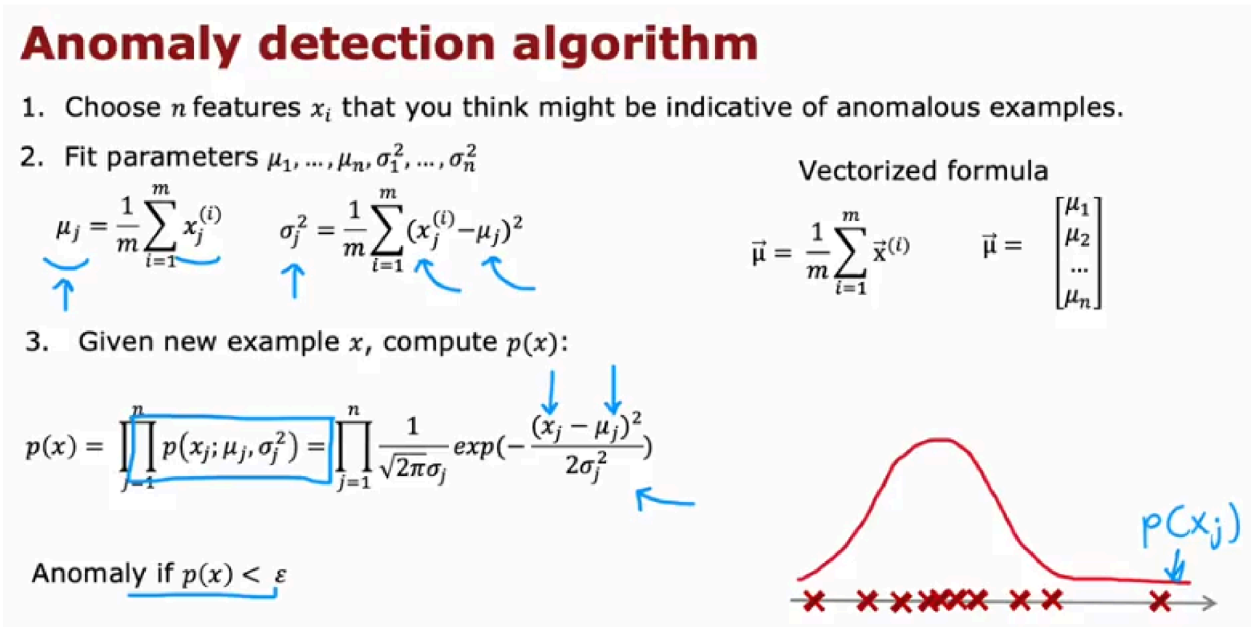
$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

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

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

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

$p(x_1 = \text{high temp}) = 1/10$
$p(x_2 = \text{high vibra}) = 1/20$
$p(x_1, x_2) = p(x_1) * p(x_2)$
$= \frac{1}{10} \times \frac{1}{20} = \frac{1}{200}$



Anomaly detection algorithm

1. Choose n features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

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

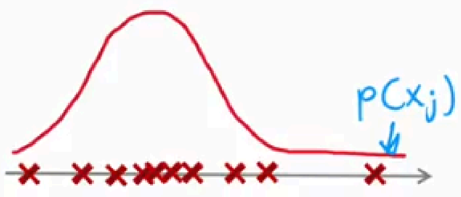
Vectorized formula

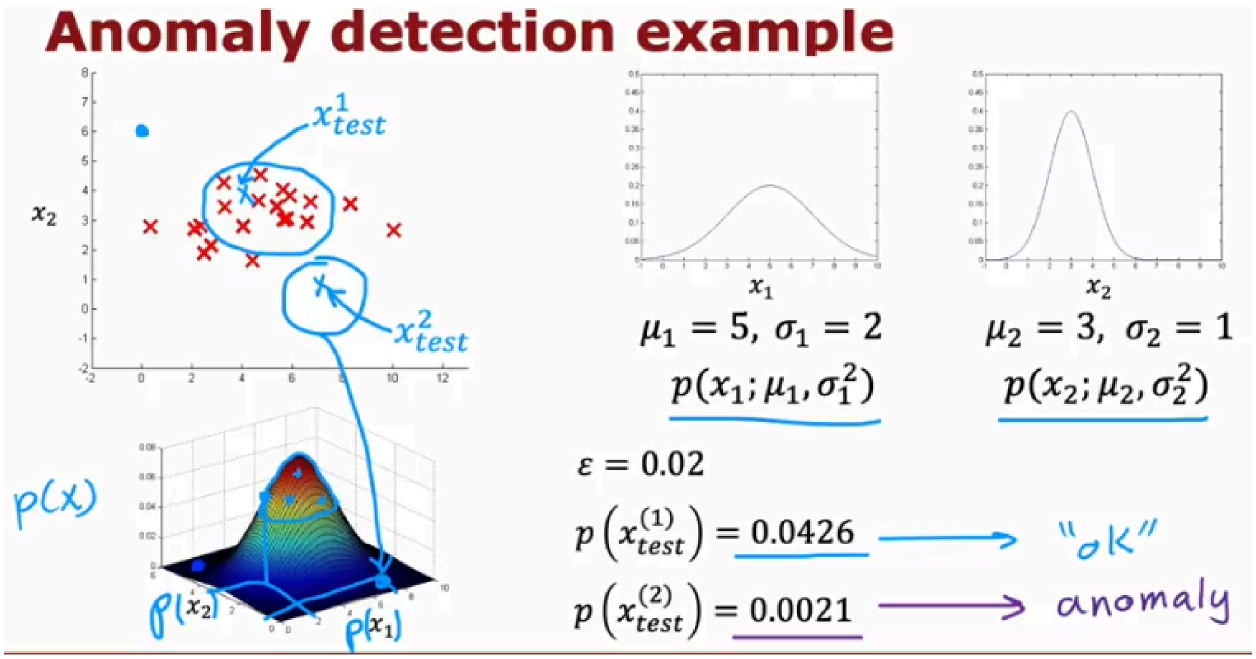
$$\vec{\mu} = \frac{1}{m}\sum_{i=1}^{m} \vec{x}^{(i)} \qquad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_n \end{bmatrix}$$

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

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

Anomaly if $\underline{p(x) < \varepsilon}$





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.

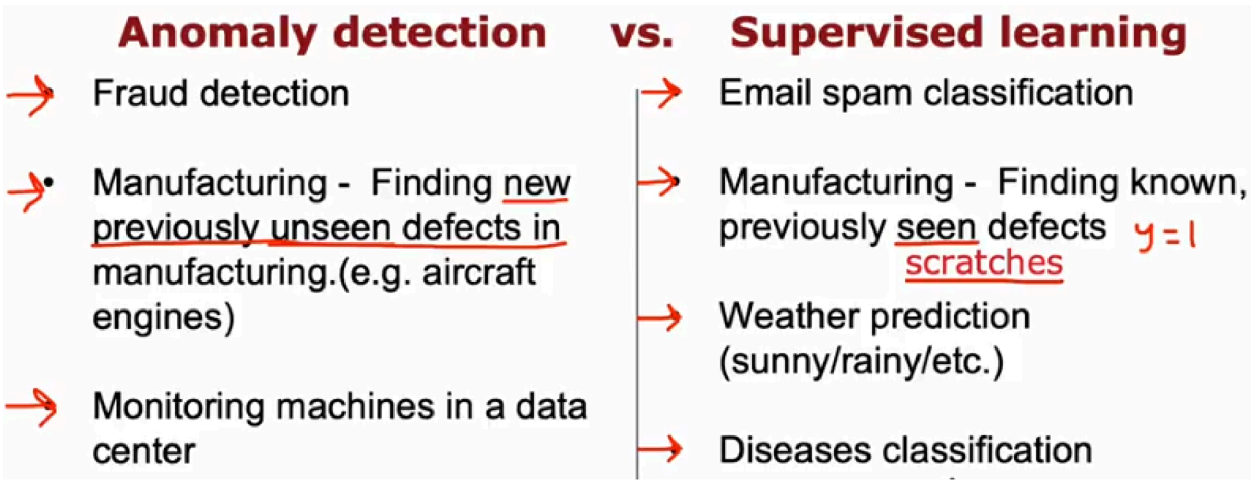
5. **Labeled Data:** Having a small number of labeled anomalies helps tune the system more effectively. The algorithm is still unsupervised, with labeled data used only for evaluation and tuning.

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.

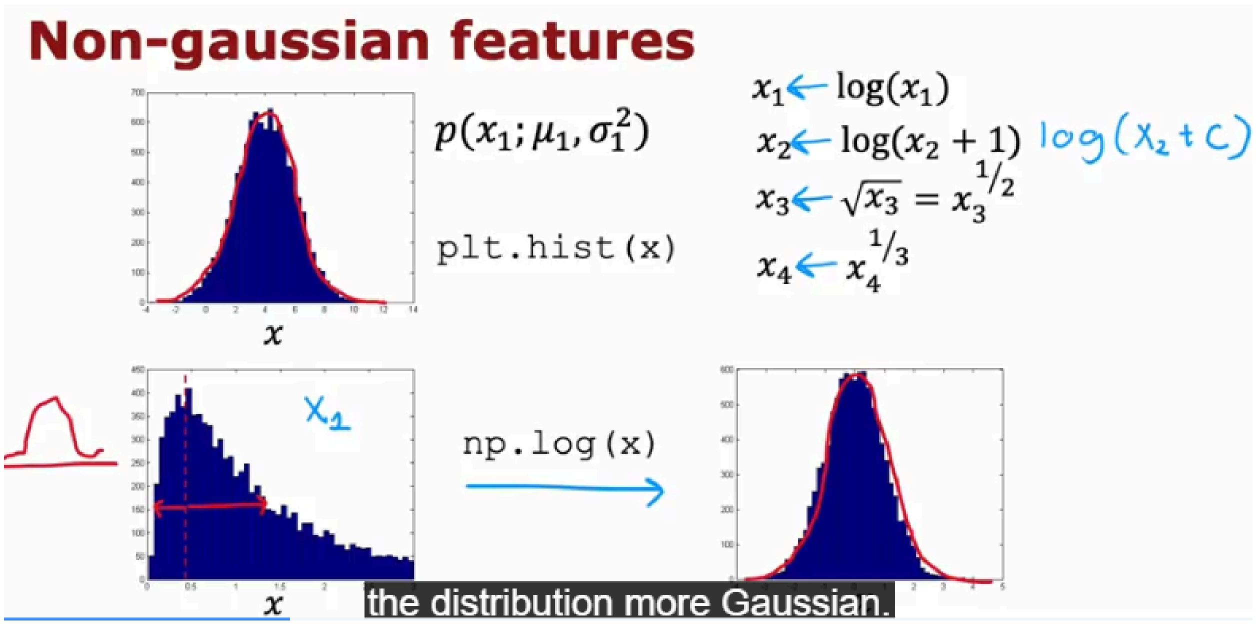


Anomaly detection vs. **Supervised learning**

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

$p(x_1; \mu_1, \sigma_1^2)$

`plt.hist(x)`

$x_1 \leftarrow \log(x_1)$

$x_2 \leftarrow \log(x_2 + 1)$ $\log(x_2 + c)$

$x_3 \leftarrow \sqrt{x_3} = x_3^{1/2}$

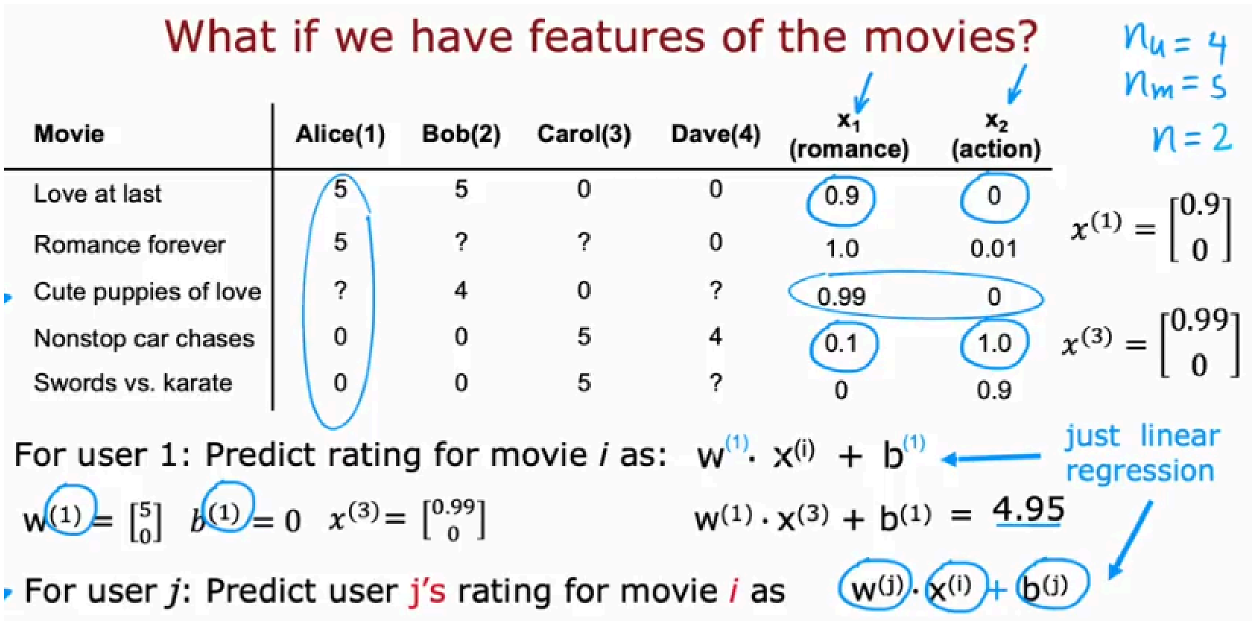
$x_4 \leftarrow x_4^{1/3}$

`np.log(x)`

the distribution more Gaussian.

Week - 2: Unsupervised Learning (Recomenders)

This lecture is about **developing a recommender system using features of items (in this case, movies)**. The approach discussed is essentially a form of **content-based filtering**, where predictions are made based on the **features of the items and preferences of users**.



What if we have features of the movies?

$n_u = 4$

$n_m = 5$

$n = 2$

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

$x^{(1)} = \begin{bmatrix} 0.9 \\ 0 \end{bmatrix}$

$x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$

For user 1: Predict rating for movie *i* as: $w^{(1)} \cdot x^{(i)} + b^{(1)}$ ← just linear regression

$w^{(1)} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ $b^{(1)} = 0$ $x^{(3)} = \begin{bmatrix} 0.99 \\ 0 \end{bmatrix}$ $w^{(1)} \cdot x^{(3)} + b^{(1)} = \underline{4.95}$

For user *j*: Predict user j's rating for movie *i* as $w^{(j)} \cdot x^{(i)} + b^{(j)}$

🔶 1. Basic Idea:

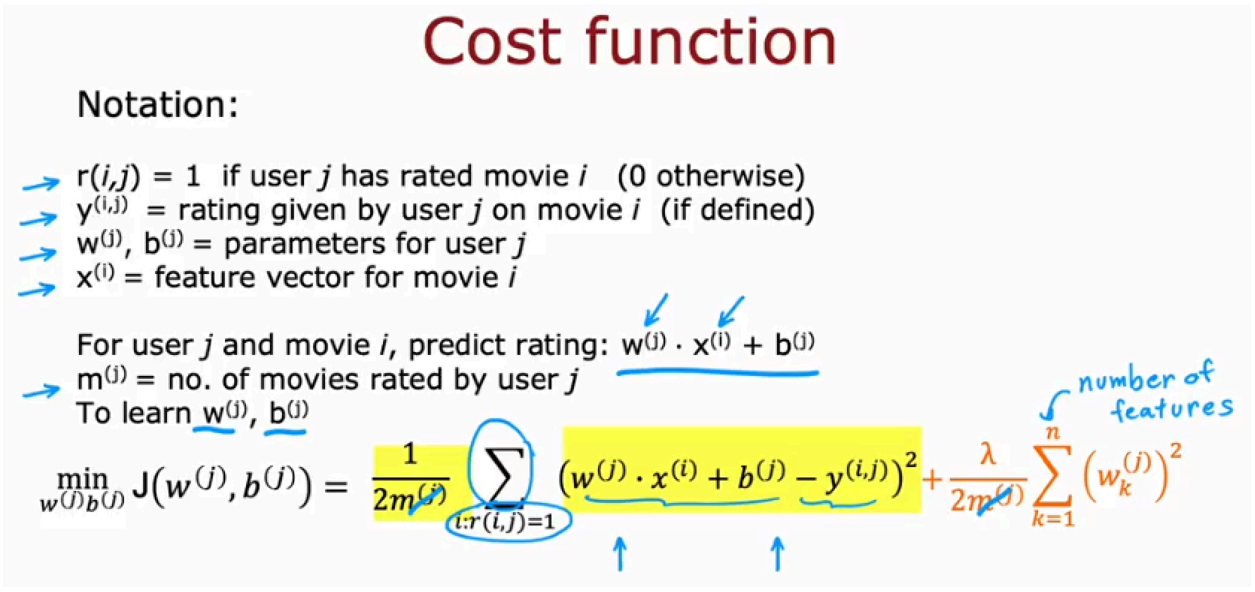
- You have **users** who rated **movies**, but not all movies are rated by every user.

- In addition to ratings, each movie also has **features** like:
    - **X1** = how romantic the movie is
    - **X2** = how action-packed the movie is

Example:

- "Love at Last" → X = [0.9, 0] (very romantic, not action)
- "Nonstop Car Chases" → X = [0.1, 1.0] (little romance, very action)

Cost Function



Cost function

Notation:

- $r(i,j) = 1$ if user j has rated movie i (0 otherwise)
- $y^{(i,j)}$ = rating given by user j on movie i (if defined)
- $w^{(j)}, b^{(j)}$ = parameters for user j
- $x^{(i)}$ = feature vector for movie i

For user j and movie i, predict rating: $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$m^{(j)}$ = no. of movies rated by user j

To learn $w^{(j)}, b^{(j)}$

$$\min_{w^{(j)} b^{(j)}} J(w^{(j)}, b^{(j)}) = \frac{1}{2\cancel{m^{(j)}}} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2\cancel{m^{(j)}}} \sum_{k=1}^{n} \left(w_k^{(j)}\right)^2$$

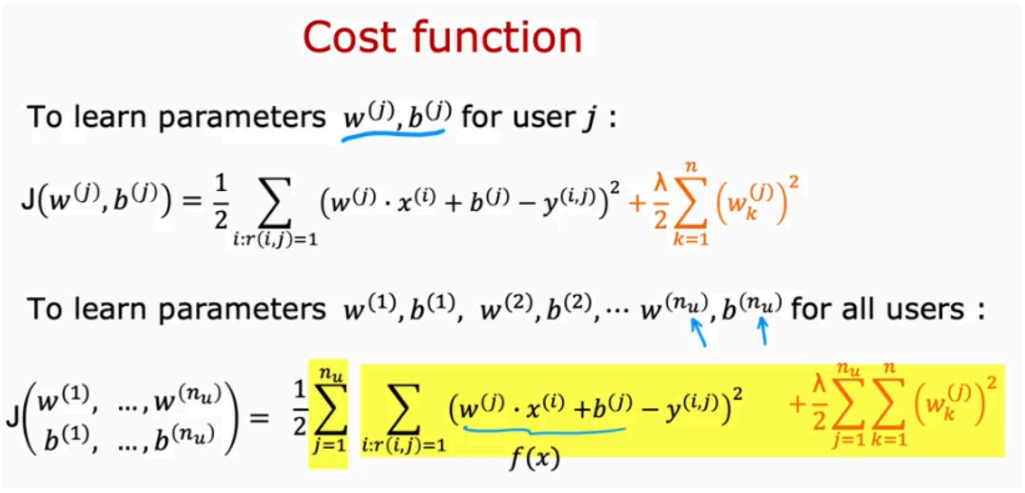
We want to find w(j) and b(j) that make our predictions as close as possible to the actual ratings.

So, we use a **Mean Squared Error (MSE)** loss:

Where:

- The sum is over all movies i rated by user j (i.e., where r(i,j) = 1)
- The second term is a **regularization** term to avoid overfitting
- λ is the regularization parameter

Now instead of training just for one user, do it for **all users**:



Cost function

To learn parameters $\underline{w^{(j)}, b^{(j)}}$ for user j :

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2}\sum_{i:r(i,j)=1}\left(w^{(j)}\cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2$$

To learn parameters $w^{(1)}, b^{(1)},\ w^{(2)}, b^{(2)}, \dots\ w^{(n_u)}, b^{(n_u)}$ for all users :

$$J\begin{pmatrix} w^{(1)}, & \dots, w^{(n_u)} \\ b^{(1)}, & \dots, b^{(n_u)} \end{pmatrix} = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left(\underbrace{w^{(j)}\cdot x^{(i)} + b^{(j)}}_{f(x)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2$$

We can now minimize this total cost using **gradient descent** (or other optimization algorithms), and learn the best `w(j)` and `b(j)` for every user.

Collaborative Filtering (Learning Features Automatically)

✅ What We Already Know:

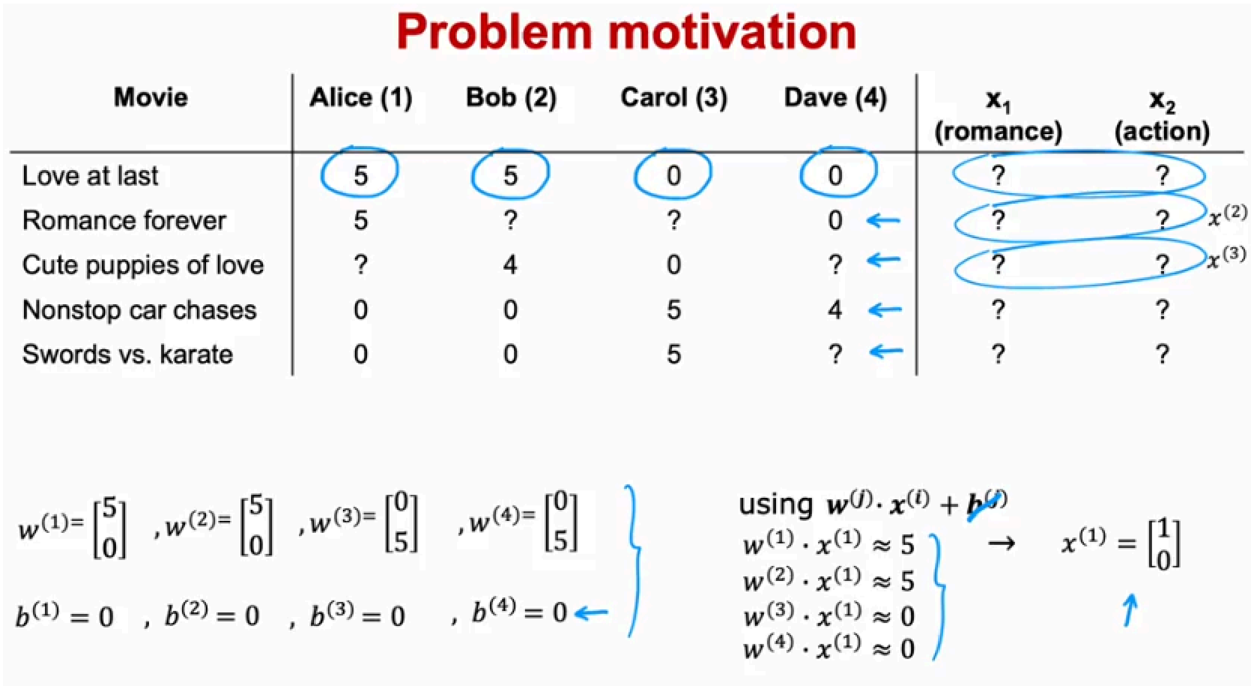
- Previously, movie features (e.g., romance, action) were **manually defined**.
- User preferences (w) + movie features (x) + bias (b) → predicted rating.

🤔 New Problem: What if Features (x) Are Unknown?

- We don't know movie features in advance.
- But we **assume we already know** user preferences (w) and biases (b).

🧠 Idea: Learn Movie Features (x) from Ratings

- Use observed ratings to **guess** feature values (x) that match user preferences.
- For each movie, minimize the **squared error** between predicted and actual ratings.



Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	? $x^{(2)}$
Cute puppies of love	?	4	0	?	?	? $x^{(3)}$
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

$w^{(1)}=\begin{bmatrix}5\\0\end{bmatrix}$, $w^{(2)}=\begin{bmatrix}5\\0\end{bmatrix}$, $w^{(3)}=\begin{bmatrix}0\\5\end{bmatrix}$, $w^{(4)}=\begin{bmatrix}0\\5\end{bmatrix}$

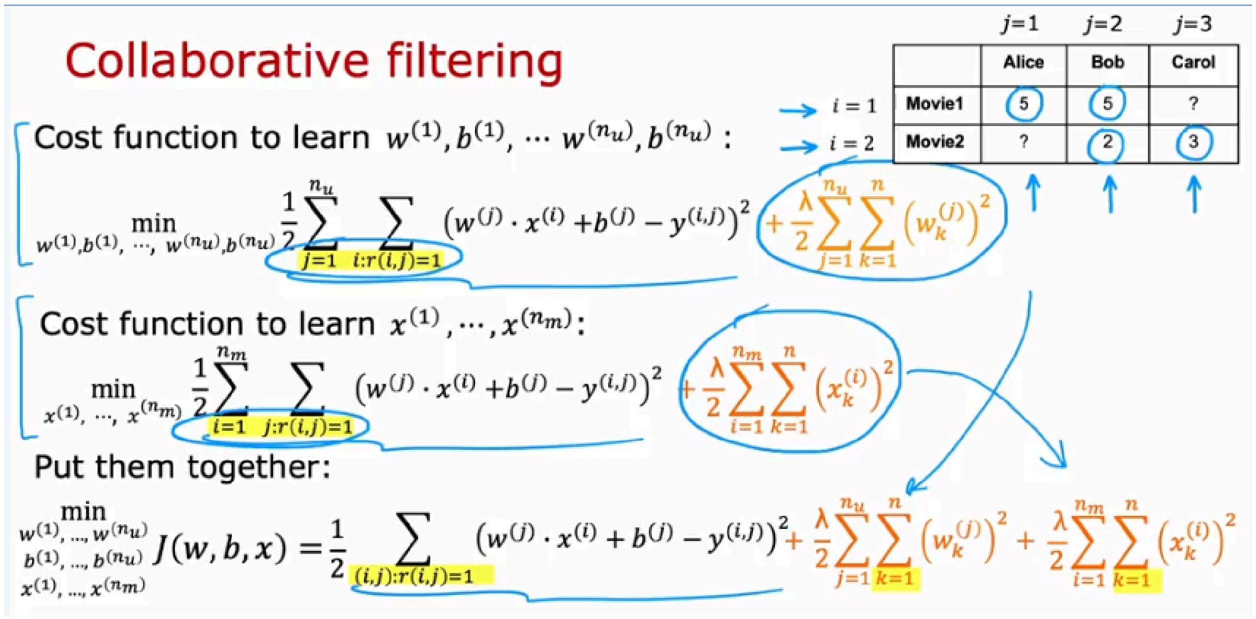
$b^{(1)}=0$, $b^{(2)}=0$, $b^{(3)}=0$, $b^{(4)}=0 \leftarrow$

using $w^{(j)}\cdot x^{(i)} + \cancel{b^{(j)}}$

$$\left.\begin{aligned} w^{(1)}\cdot x^{(1)} &\approx 5\\ w^{(2)}\cdot x^{(1)} &\approx 5\\ w^{(3)}\cdot x^{(1)} &\approx 0\\ w^{(4)}\cdot x^{(1)} &\approx 0 \end{aligned}\right\} \rightarrow \quad x^{(1)}=\begin{bmatrix}1\\0\end{bmatrix}$$

🔄 Learning Both Users and Movies

- In practice, we **don't know** either `w`, `b`, or `x`.
- So, we **jointly learn**:
  - Movie features `x`^{i}
  - User preferences `w`^{j}
  - User bias `b`^{j}



Collaborative filtering

		$j=1$ Alice	$j=2$ Bob	$j=3$ Carol
$i=1$	Movie1	5	5	?
$i=2$	Movie2	?	2	3

Cost function to learn $w^{(1)}, b^{(1)}, \cdots w^{(n_u)}, b^{(n_u)}$:

$$\min_{w^{(1)},b^{(1)},\dots,w^{(n_u)},b^{(n_u)}} \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}\left(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2$$

Cost function to learn $x^{(1)}, \cdots, x^{(n_m)}$:

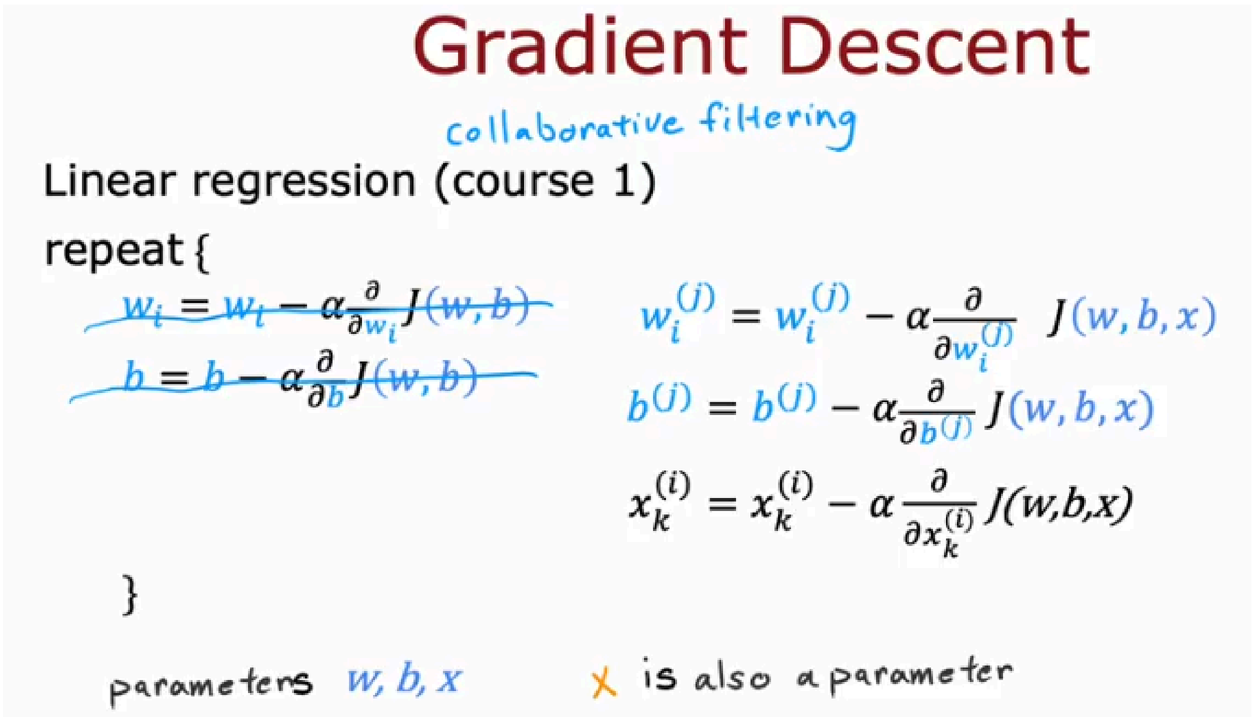
$$\min_{x^{(1)},\dots,x^{(n_m)}} \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)=1}\left(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2$$

Put them together:

$$\min_{\substack{w^{(1)},\dots,w^{(n_u)}\\ b^{(1)},\dots,b^{(n_u)}\\ x^{(1)},\dots,x^{(n_m)}}} J(w,b,x) = \frac{1}{2}\sum_{(i,j):r(i,j)=1}\left(w^{(j)}\cdot x^{(i)}+b^{(j)}-y^{(i,j)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}\left(w_k^{(j)}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2$$

⚙️ Optimization: Use Gradient Descent

- Update w, b, and x iteratively to minimize the cost.
- Learn everything from the data — no hand-crafted features needed.



Gradient Descent

collaborative filtering

Linear regression (course 1)

repeat {

~~$w_i = w_i - \alpha \frac{\partial}{\partial w_i} J(w,b)$~~

~~$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$~~

$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w,b,x)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w,b,x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial x_k^{(i)}} J(w,b,x)$$

}

parameters w, b, x

x is also a parameter

✨ Key Insight:

- Ratings from **multiple users** of the same movie allow us to **collaboratively** infer features.
- This is why it's called **Collaborative Filtering**.

The main goal is to generalize the collaborative filtering algorithm, typically used for predicting ratings, to work with binary labels, which are simpler and reflect user engagement more directly.

Here's a breakdown of the process:

1. **Binary Labels**:
   - **1**: User liked or engaged with the item (e.g., watched a movie, clicked on a product).
   - **0**: User did not engage with the item (e.g., skipped a movie, ignored a product).
   - **?**: User has not yet been exposed to the item.
2. **Logistic Regression**:
   - The model predicts the probability of user engagement using the logistic function:

$$P(y_{ij} = 1) = g(w_j \cdot x_i + b)$$

   - This transforms the prediction from a continuous value to a probability between 0 and 1.
3. **Cost Function**:
   - The binary cross-entropy loss function is used:

$$\text{Loss} = -y \log(f(x)) - (1 - y) \log(1 - f(x))$$

   - The cost function is summed over all user-item pairs where ratings exist.

Adapting the Algorithm:

- Previously, the algorithm used a linear regression model where the prediction was in the form $y_{ij} = w_j \cdot x_i + b$. This would predict a numerical rating.
- To work with binary labels, we shift to **logistic regression**, which predicts probabilities. The formula becomes:

$$P(y_{ij} = 1) = g(w_j \cdot x_i + b)$$

  where $g(z) = \frac{1}{1+e^{-z}}$ is the **logistic function** (also known as the sigmoid function).
- This transforms the prediction from a continuous value to a probability between 0 and 1, indicating the likelihood that a user will engage with the item.

5. **Application in Various Domains:**
   - **E-commerce:** Whether a user purchases a product after being shown it (binary 1 or 0).
   - **Social Media:** Whether a user likes or favorites a post after being shown it.
   - **Advertising:** Whether a user clicks on an ad after seeing it.
6. **Generalization:**
   - By using the logistic function and binary cross-entropy loss, we generalize collaborative filtering to handle binary interactions, which is suitable for many real-world applications like recommending products, movies, or ads based on user engagement.

Mean normalization in collaborative filtering

🔧 Problem Without Mean Normalization

- Collaborative filtering uses parameters `w` and `b` for each user.
- For a new user (like Eve) who hasn't rated any movies:
  - Their parameters will be initialized as `w = [0, 0]` and `b = 0`.
  - The model predicts all ratings as `0`, which is unrealistic.

🧠 What is Mean Normalization?

- For each **movie**, calculate its **average rating** (e.g., 2.5 stars).
- Subtract this average (mean) from each user’s rating for that movie.
- This shifts all movie ratings to have a mean of **0**.
- These adjusted ratings are used for training the model.

📈 Prediction with Mean Normalization

- During training: Model predicts `w(j) = x(i) + b(j)` using normalized ratings.