## **📘 Unsupervised Learning - Lecture Note**

### **What is Unsupervised Learning?**

Unsupervised learning is a type of machine learning where the algorithm learns patterns from **unlabeled data**.  
 That means we **do not provide any output labels**—the model tries to discover hidden patterns, groupings, or structure from the input data by itself.

🔑 **Key Idea:** The model explores the data and identifies natural patterns without being explicitly told what to look for.

### **Why is Unsupervised Learning Important?**

* **Real-world data is often unlabeled.**
* **Labeling is time-consuming and expensive.**
* Helps in discovering **hidden insights** and **relationships** within data.
* Essential in **data exploration**, **pattern recognition**, **anomaly detection**, and **feature learning**.

### **Common Tasks in Unsupervised Learning**

| **Task** | **Description** | **Example** |
| --- | --- | --- |
| **Clustering** | Group similar data points | Grouping customers by behavior |
| **Dimensionality Reduction** | Reduce number of features while keeping important info | Visualizing high-dimensional data in 2D |
| **Association Rule Learning** | Discover rules between features | “People who buy X also buy Y” |
| **Anomaly Detection** | Detect outliers or unusual data points | Fraud detection in finance |

### **1. Clustering**

#### **What it is:**

Divides data into groups (clusters) where items in the same group are more similar to each other than to those in other groups.

#### **🧪 Algorithms:**

* **K-Means Clustering**
* **Hierarchical Clustering**
* **DBSCAN**

#### **🔍 Use Cases:**

* Customer segmentation
* Market research
* Image grouping
* Document clustering

### **2. Dimensionality Reduction**

#### **📌 What it is:**

Reduces the number of input variables in a dataset while retaining most of the information.

#### **🧪 Techniques:**

* **PCA (Principal Component Analysis)**
* **t-SNE (t-distributed Stochastic Neighbor Embedding)**
* **UMAP (Uniform Manifold Approximation and Projection)**

#### **🔍 Use Cases:**

* Data visualization
* Noise reduction
* Preprocessing before applying other models

### **3. Association Rule Mining**

#### **📌 What it is:**

Finds interesting relationships (associations) between variables in large datasets.

#### **🧪 Techniques:**

* **Apriori Algorithm**
* **FP-Growth Algorithm**

#### **🔍 Use Cases:**

* Market basket analysis (e.g., “If a customer buys bread, they also buy butter”)
* Recommender systems

### **4. Anomaly Detection**

#### **📌 What it is:**

Identifies rare events or outliers in the dataset that do not conform to expected patterns.

#### **🧪 Techniques:**

* **Isolation Forest**
* **One-Class SVM**
* **Clustering-based outlier detection**

#### **🔍 Use Cases:**

* Fraud detection
* Network security
* Fault detection in machines

### **🔧 Popular Algorithms in Unsupervised Learning**

| **Algorithm** | **Type** | **Best For** |
| --- | --- | --- |
| **K-Means** | Clustering | Grouping data into clusters |
| **DBSCAN** | Clustering | Discovering clusters of varying shapes |
| **PCA** | Dimensionality Reduction | Data compression, visualization |
| **t-SNE** | Dimensionality Reduction | High-dimensional data visualization |
| **Apriori** | Association Rules | Frequent itemset mining |
| **Isolation Forest** | Anomaly Detection | Fraud, intrusion detection |

### **📁 Example Datasets for Practice**

| **Dataset** | **Description** |
| --- | --- |
| **Iris Dataset** | Flower features – perfect for clustering |
| **MNIST** | Handwritten digits – image clustering |
| **MovieLens** | User ratings – recommender systems |
| **Mall Customers** | Customer demographics & spending score |
| **Online Retail Dataset** | Purchase transactions for association mining |

### **📎 Benefits of Unsupervised Learning**

* No need for labeled data
* Discovers unknown patterns and relationships
* Great for exploratory data analysis

### **⚠️ Challenges of Unsupervised Learning**

* No clear way to evaluate accuracy (no labels)
* Choosing the right algorithm and parameters can be hard
* Results may not always be interpretable

### **🎓 Hands-on Project Ideas for Students**

1. **Customer Segmentation using K-Means**
2. **Product Recommendations using Association Rules**
3. **Topic Modeling in News Articles**
4. **Anomaly Detection in Credit Card Transactions**
5. **Clustering Images from the MNIST Dataset**

### **✅ Key Takeaways**

* Unsupervised learning **doesn’t need labeled data**.
* It’s useful for exploring data and discovering hidden structures.
* Core tasks include **clustering**, **dimensionality reduction**, **association mining**, and **anomaly detection**.
* Practical applications range from marketing to fraud detection.

**Lesson 2**

# **Clustering in Machine Learning – Lecture Notes**

## **What is Clustering?**

Clustering is an *unsupervised learning* technique used to group similar data points together without predefined labels. It helps discover hidden patterns or groupings in data.

🔑 Key Idea: Group data points that are similar to each other into *clusters*.

## **Why Use Clustering?**

**Clustering is used in:**

* **Customer segmentation (e.g., marketing)**
* **Document classification**
* **Anomaly detection**
* **Image compression**
* **Social network analysis**

## **💡 Common Clustering Algorithms**

### **1. K-Means Clustering**

* **One of the simplest and most popular algorithms.**
* **Objective: Minimize the distance between data points and their cluster centroids.**

#### **Steps:**

1. **Choose number of clusters K**
2. **Randomly initialize K centroids.**
3. **Assign each point to the nearest centroid.**
4. **Recalculate centroids based on assigned points.**
5. **Repeat steps 3–4 until convergence.**

#### **Pros:**

* **Simple and fast.**
* **Works well with spherical clusters.**

#### **Cons:**

* **Needs predefined K.**
* **Sensitive to outliers.**
* **Not good for non-spherical clusters.**

### **2. Hierarchical Clustering**

* **Builds a hierarchy of clusters.**
* **Doesn’t require you to specify K beforehand.**

#### **Types:**

* **Agglomerative (Bottom-Up): Start with each point as its own cluster, then merge.**
* **Divisive (Top-Down): Start with one cluster, then split.**

#### **Output:**

* **Dendrogram: A tree-like diagram showing merges/splits.**

### **3. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

* **Groups points that are close together based on density.**
* **Can find arbitrarily shaped clusters and identify noise (outliers).**

#### **Parameters:**

* **ε (epsilon) – radius of neighborhood.**
* **MinPts – minimum number of points to form a dense region.**

#### **Pros:**

* **Detects noise.**
* **Handles arbitrary shapes.**

#### **Cons:**

* **Struggles with varying densities.**
* **Needs good tuning of ε and MinPts.**

## **📊 Visual Example**

**Imagine plotting the following points:**

**less**

**CopyEdit**

**Cluster 1: [1,2], [2,2], [1.5,1.8]**

**Cluster 2: [8,8], [9,8], [8.5,8.2]**

**K-Means will assign them to separate centroids, while DBSCAN might identify a dense region around each group.**

## **📐 Distance Metrics Used**

* **Euclidean Distance: Most common, for geometric distances.**
* **Manhattan Distance: Sum of absolute differences.**
* **Cosine Similarity: For high-dimensional data like text.**

## **🧪 Evaluation Metrics for Clustering**

**Since we don’t have labels (unsupervised), evaluation can be tricky.**

### **Internal Metrics:**

* **Silhouette Score: Measures cohesion & separation (-1 to +1).**
* **Davies-Bouldin Index: Lower is better.**

### **External Metrics (If you have labels for evaluation):**

* **Adjusted Rand Index**
* **Normalized Mutual Information**

## **Applications of Clustering**

| **Application** | **Description** |
| --- | --- |
| **Market Segmentation** | **Group customers based on buying behavior** |
| **Image Segmentation** | **Separate objects in an image** |
| **Document Clustering** | **Group similar articles** |
| **Fraud Detection** | **Spot unusual patterns as outliers** |

# **K-Means Clustering: Deep Dive**

## **Real-Life Example: Customer Segmentation for a Supermarket**

**Imagine you're running a supermarket and have the following data on your customers:**

* **Age**
* **Annual income**
* **Amount spent monthly**

**You want to group these customers into segments to:**

* **Send personalized marketing campaigns**
* **Offer loyalty discounts to high spenders**
* **Identify budget-conscious shoppers**

**But you don’t know beforehand how many types of customers you have — that’s where K-Means comes in.**

## **How K-Means Works**

### **Step-by-step Breakdown:**

1. **Choose the number of clusters K (this is a hyperparameter).**
2. **Randomly initialize K centroids in the feature space.**
3. **Assign each data point to the nearest centroid using a distance metric (commonly Euclidean).**
4. **Recalculate the centroids by taking the mean of all points in each cluster.**
5. **Repeat steps 3–4 until convergence (i.e., centroids no longer move or max iterations reached).**

## **Visualization of the Process**

**Let’s say we have customers plotted based on:**

* **X-axis: Age**
* **Y-axis: Monthly Spend**

### **Iteration 1:**

* **Randomly place 3 centroids (red Xs)**
* **Assign customers to closest centroid → clusters form**

### **Iteration 2:**

* **Recalculate centroids as the mean of assigned points**
* **Reassign customers**
* **Repeat...**

**After a few iterations, the clusters stabilize.**

## **How Are Data Points Clustered?**

**Clustering is based on distance to centroids.**

**Example (Euclidean Distance):**

**distance=(x2−x1)2+(y2−y1)2\text{distance} = \sqrt{(x\_2 - x\_1)^2 + (y\_2 - y\_1)^2}distance=(x2​−x1​)2+(y2​−y1​)2​**

**Each data point is assigned to the cluster whose centroid is nearest.**

**Under the Hood in Machine Learning**

**K-Means optimizes the within-cluster sum of squares (WCSS):**

**WCSS=∑i=1K∑x∈Ci∥x−μi∥2\text{WCSS} = \sum\_{i=1}^K \sum\_{x \in C\_i} \| x - \mu\_i \|^2WCSS=i=1∑K​x∈Ci​∑​∥x−μi​∥2**

**Where:**

* **CiC\_iCi​ is cluster i**
* **μi\mu\_iμi​ is the centroid of cluster i**
* **xxx is a data point**

**Goal: Minimize WCSS → tighter clusters.**

## **Model Evaluation Metrics**

**Since K-Means is unsupervised, you don’t have true labels to compare. So we use internal evaluation metrics.**

### **1. Elbow Method (for selecting K)**

* **Plot WCSS vs. K**
* **Look for the "elbow" point (where WCSS decreases sharply then levels off)**
* **That’s your optimal K**

### **Problem Statement**

**You're given a Mall Customer dataset containing the following columns:**

* **CustomerID: Unique ID for each customer**
* **Gender: Male or Female**
* **Age: Age of the customer**
* **Annual Income (k$): Customer’s yearly income in thousands**
* **Spending Score (1-100): Score assigned by the mall based on customer spending behavior**

**Objective:  
 Segment customers into different groups based on their spending patterns and income levels using KMeans clustering. These segments can help the business with targeted marketing strategies.**

**Coding**

wcss = [] # Within-cluster sum of squares

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

kmeans.fit(X\_scaled)

wcss.append(kmeans.inertia\_)

plt.plot(range(1, 11), wcss, marker='o')

plt.title('The Elbow Method')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS')

plt.show()

**Explanation to the code**

**wcss = []**  # Within-cluster sum of squares

* wcss stands for **Within-Cluster Sum of Squares**.
* It measures how tight the clusters are (i.e., how close the data points are to their respective cluster centers).
* Lower WCSS means the clusters are compact and well-separated.

**for i in range(1, 11):**

* This loop tries different numbers of clusters from **1 to 10**.
* We do this to see how WCSS changes as we increase the number of clusters.

**kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)**

* For each i, create a KMeans model with i clusters.
* init='k-means++' initializes centroids in a smart way to speed up convergence.
* random\_state=42 ensures reproducibility (same result every time you run).

**kmeans.fit(X\_scaled)**

* Fit the KMeans model on the scaled data (X\_scaled).
* The model finds i cluster centers and assigns each point to the nearest cluster.

**wcss.append(kmeans.inertia\_)**

* kmeans.inertia\_ is the total WCSS for that number of clusters.
* We save this value in the wcss list to track how WCSS changes with more clusters.

**plt.plot(range(1, 11), wcss, marker='o')**

**plt.title('The Elbow Method')**

**plt.xlabel('Number of Clusters')**

**plt.ylabel('WCSS')**

**plt.show()**

Plot the WCSS values against the number of clusters.

Look for the "elbow point" – the point where adding more clusters doesn’t significantly reduce WCSS.

That elbow indicates a good number of clusters to use (often 3–5 in practice).

# Let's say optimal clusters = 5

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state=42)

y\_kmeans = kmeans.fit\_predict(x\_scaled)

# Add cluster to original dataframe

customer\_data['Cluster'] = y\_kmeans

# Visualize clusters

plt.figure(figsize=(8, 6))

sns.scatterplot(x=x\_scaled[:, 0], y=x\_scaled[:, 1], hue=customer\_data['Cluster'], palette='tab10', s=100)

plt.title('Customer Segments')

plt.xlabel('Annual Income (scaled)')

plt.ylabel('Spending Score (scaled)')

plt.show()