**AI Roadmap—Unsupervised Learning**

We will be learning:

|  |  |  |
| --- | --- | --- |
| Clustering, Dimensionality Reduction | Scikit-learn | Cluster customer data, PCA for image compression |

* Dimensionality reduction in machine learning is the process of reducing the number of input features (dimensions) in a dataset while preserving as much relevant information as possible. It helps improve model performance, reduces overfitting, and makes data visualization easier.
* **Input features** are the individual measurable properties or variables used to describe the data in a machine learning model. They are the inputs that the model uses to make predictions or decisions.
* **machine learning category**, with examples:
* **🔹 1. Supervised Learning**
* **➤ a. Regression**
* **Goal**: Predict a **continuous numeric** value.
* **Examples**:
* Predict stock prices
* Estimate temperature
* Predict car mileage
* **➤ b. Classification**
* **Goal**: Predict a **class/category**.
* **Examples**:
* Email: Spam or Not Spam
* Tumor: Malignant or Benign
* Image: Cat, Dog, or Horse
* **🔹 2. Unsupervised Learning**
* **➤ a. Clustering**
* **Goal**: Group similar items together without labels.
* **Examples**:
* Grouping customers by purchase behavior
* Market segmentation
* Document clustering (e.g., news articles)
* **➤ b. Dimensionality Reduction**
* **Goal**: Reduce number of features while keeping important patterns.
* **Examples**:
* PCA (Principal Component Analysis)
* t-SNE / UMAP for data visualization
* Noise reduction in image processing
* **➤ c. Anomaly Detection**
* **Goal**: Find unusual data points.
* **Examples**:
* Fraud detection
* Network intrusion detection
* Fault detection in machinery
* **🔹 3. Semi-Supervised Learning**
* **Mix of labeled + unlabeled data**
* Used when labeling data is expensive or time-consuming.
* **Examples**:
* Face recognition with only a few labeled photos
* Text classification with few tagged documents
* **🔹 4. Reinforcement Learning**
* **➤ a. Value-Based Methods**
* **Learn value of each action** in a given state.
* **Example**: Q-Learning
* **➤ b. Policy-Based Methods**
* **Learn the best action (policy)** directly.
* **Example**: REINFORCE algorithm
* **➤ c. Actor-Critic Methods**
* Combine value + policy learning.
* **Example**: Deep Deterministic Policy Gradient (DDPG)
* **Use Cases**:
* Game AI (e.g., AlphaGo)
* Robotics (e.g., robot arm learning to grasp)
* Self-driving cars (e.g., learning to navigate roads)
* **🧠 Bonus: Other Learning Types (less common but useful)**

| * **Type** | * **Description** | * **Example** |
| --- | --- | --- |
| * Self-Supervised Learning | * Learns to predict part of input from other parts | * Language models like GPT, BERT |
| * Online Learning | * Learns continuously as new data arrives | * Real-time stock predictions |
| * Multi-Task Learning | * Learns multiple tasks at once | * Predicting age and gender from face images |

* **What is PCA (Principal Component Analysis)?**
* **PCA** is a **dimensionality reduction** technique used in machine learning and data analysis. It helps simplify complex datasets by reducing the number of features (columns) while keeping the most important information.
* **🔍 In Simple Terms:**
* PCA **finds new axes (called principal components)** that:
* Are combinations of your original features
* Capture the **most variance** (spread) in your data
* Are **uncorrelated** with each other
* Let you **project high-dimensional data into fewer dimensions**
* It uses covariance matrix, eigen vectors, eigen values
* StandardScaler() transforms each feature by subtracting its mean and dividing by its standard deviation, so the data has a mean of 0 and a standard deviation of 1. This puts all features on the same scale, which helps many machine learning algorithms perform better and converge faster.
* **Sklearn functions** for each scaling type:
* **1. Standardization (StandardScaler)**
* Scales data to have **mean = 0** and **standard deviation = 1**
* Formula: (x−mean)/std(x - \text{mean}) / \text{std}(x−mean)/std
* Works well if data is normally distributed
* Used by algorithms like SVM, Logistic Regression, KNN
* 🛠️ **sklearn function**: sklearn.preprocessing.StandardScaler
* **2. Min-Max Scaling (Normalization)**
* Scales data to a fixed range, usually **0 to 1**
* Formula: (x−min)/(max−min)(x - \text{min}) / (\text{max} - \text{min})(x−min)/(max−min)
* Preserves the shape of original distribution
* Useful when you need features on the same scale but want to keep original distribution range
* 🛠️ **sklearn function**: sklearn.preprocessing.MinMaxScaler
* **3. MaxAbs Scaling**
* Scales data by dividing by the **maximum absolute value** of each feature
* Scales data to range between **-1 and 1** without shifting the center
* Useful for data that’s already centered at zero but needs scaling
* 🛠️ **sklearn function**: sklearn.preprocessing.MaxAbsScaler
* **4. Robust Scaling**
* Uses **median** and **Interquartile Range (IQR)** instead of mean/std
* Formula: (x−median)/IQR(x - \text{median}) / \text{IQR}(x−median)/IQR
* Robust to outliers, good when data has extreme values
* 🛠️ **sklearn function**: sklearn.preprocessing.RobustScaler
* **5. Logarithmic Scaling**
* Applies **log transformation** to compress large values and reduce skewness
* Useful when data spans several orders of magnitude or is heavily skewed
* 🛠️ **Note**: No direct scaler in sklearn; usually done with numpy.log or pandas.apply(np.log)
* **6. Unit Vector Scaling (Normalization)**
* Scales each sample (row) to have unit norm (length = 1)
* Useful in text classification and clustering (e.g., TF-IDF vectors)
* 🛠️ **sklearn function**: sklearn.preprocessing.Normalizer
* **random\_state** is a **seed value** used to **control randomness** in functions that involve random processes — like splitting data, initializing centroids (in KMeans), or shuffling.

### 🔍 Why it matters:

* Many ML functions (like train\_test\_split, KMeans, etc.) involve randomness.  
  If you don’t fix the seed, you’ll get **different results every time** you run the code.
* By setting random\_state, you make the process **reproducible**.
* random\_state ensures **consistent, repeatable results** when randomness is involved.
* **KMeans** is an unsupervised machine learning algorithm used for **clustering**, which groups similar data points into **k clusters** based on their features. It starts by randomly selecting **k initial centroids**, then repeatedly performs two steps: assigning each data point to the nearest centroid, and updating each centroid to be the mean of the points in its cluster. This process continues until **convergence**, when the centroids no longer change significantly. KMeans only works with **numerical data**, as it relies on distance calculations, and its behavior can vary depending on the **initial centroids**. To make the results **reproducible**, a random\_state parameter is often used. While convergence may happen quickly, in some cases it takes multiple iterations, especially if centroids are initialized far from the true cluster centers.
* Let's say you have **6 data points** in 2D space:
* (1, 1), (1.5, 2), (3, 4), (5, 7), (3.5, 5), (4.5, 5)
* You want to cluster them into **k = 2** groups.
* **⚙️ Step-by-step Execution:**
* **🔹 Step 1: Random Initialization**
* Let’s say KMeans picks **initial centroids**:
* Centroid A: (1, 1)
* Centroid B: (5, 7)
* **🔹 Step 2: First Assignment**
* Calculate distance of each point to the centroids, and assign accordingly:
* Cluster A: (1,1), (1.5,2), (3,4)
* Cluster B: (5,7), (3.5,5), (4.5,5)
* **🔹 Step 3: Recalculate Centroids**
* New Centroid A: mean of (1,1), (1.5,2), (3,4) → approx (1.83, 2.33)
* New Centroid B: mean of (5,7), (3.5,5), (4.5,5) → approx (4.33, 5.67)
* **🔹 Step 4: Reassign Points**
* Now recalculate distances to **new centroids**.
* You might see:
* **Point (3,4)** moves from **Cluster A to Cluster B**
* Now:
* Cluster A: (1,1), (1.5,2)
* Cluster B: (3,4), (3.5,5), (4.5,5), (5,7)
* **🔹 Step 5: Recalculate Again**
* New Centroid A: mean of (1,1), (1.5,2) → (1.25, 1.5)
* New Centroid B: mean of the rest → approx (4, 5.25)
* **🔁 This process continues until:**
* No points change clusters
* Or centroids move very little
* In this case, **convergence may take 3–5 iterations**, depending on how much the points jump between clusters.
* **Explained Variance** is a measure of how much of the total variability (or information) in the data is captured by a model or a set of components (like in PCA).
* **In simple terms:**
* It tells you **how well your model or components represent the original data**.
* For example, in **PCA**, each principal component explains a certain amount of variance in the data.
* The **explained variance ratio** shows the percentage of total variance captured by each principal component.
* **What does the Explained Variance Plot Show?**
* **X-axis:** Number of PCA components (from 1 up to n\_components).
* **Y-axis:** Cumulative explained variance (percentage of total variance captured) up to that number of components.
* **What it represents:**
* Each PCA component captures some amount of variance (information) from the original data.
* The plot shows **how much total variance is explained if you keep the first N components**.
* The curve is typically **increasing and saturates** because adding more components explains more variance but with diminishing returns.