**Clustering**

Clustering is a fundamental task in unsupervised learning, where the goal is to **group data points based on their features** without having any predefined labels or targets.

How Clustering Works

1. Given a dataset X, clustering algorithms group data into k clusters based on feature similarity.
2. The number of clusters k may be pre-defined or determined dynamically using evaluation metrics.

**Clustering: K-Means**

Partitions data into k clusters by minimizing the distance from each point to its nearest cluster center, where each data point belongs to cluster with the nearest mean (centroid).

1. Not to be confused with k-NN:

* K-Means is unsupervised (no labels), while k-NN is supervised (uses labels for classification).

1. Distance Metric:

* K-Means uses Euclidean distance by default in sklearn to measure similarity between points and centroids.

**How K-Means Works**

* Initialization: Choose k random points as initial cluster centroids (means).
* Assignment Step (A):
  + For each data point, assign it to the nearest centroid based on Euclidean distance.
  + Fix means 🡺 assign all datapoints to their closest mean.
* Update Step (B):
  + Recalculate the centroids by taking the mean of all points assigned to each cluster.
  + Fix cluster assignment 🡺 recalculate means.
* Repeat and alternate between Assignment (A) and Update (B) until:
  + Convergence: Centroids stop changing.
  + Maximum Iterations: A limit is reached.

**Coding step**

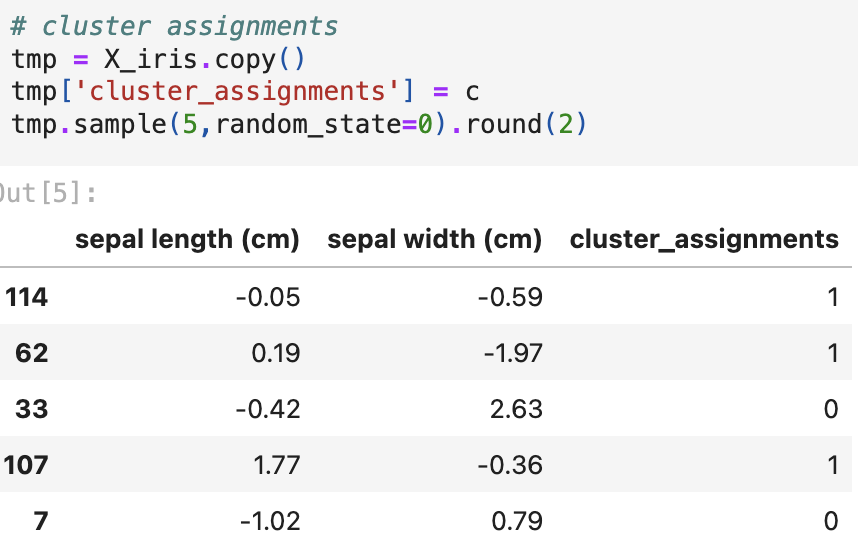
1. **Determine the # of clusters**

A math equations on a white background

Description automatically generated

* K-means algo will assign each of observations to one of 2 pre-defined clusters
* **init = ‘random’** 🡺 initial cluster centroids are selected randomly
  + the default centroids are usually defined as **k-means++**
* Fit the model and predict clusters for dataset.

1. **Clustering assignment**



* Each row contains length and width 2 features and assigned clusters.
* Datapoints with similar features are grouped into clusters, labeled as 0 and 1.

1. **Coordinates of centroid**

A close-up of numbers

Description automatically generated

* Cluster 0 centered at (-.98, .9)
* Cluster 1 centered at (.49, -.45)

1. **Centroid and cluster visualization**

A graph with blue and orange dots

Description automatically generated

**Evaluation of K-means using within cluster sum of squared distance (SSD)**

* It measures how well the K-Means algorithm has grouped the data points into clusters.
* **High SSD**:
  + Points are far from their assigned cluster centers.
  + Clusters are spread out and poorly compact.
* **Low SSD**:
  + Points are close to their assigned cluster centers.
  + Clusters are tight and well-formed.

**Parameters to determine K-Means Clustering**

1. **The # of Clusters k (n\_clusters):**

* If k is too small, clusters may merge incorrectly.
* If k is too large, clusters may over-segment the data.
* **Elbow Method**: Look for the point where SSD decreases sharply to choose k.

A graph with a line going up

Description automatically generated

1. **Initial Locations of Centroids**

* The starting cluster centroids affect the algorithm’s performance because K-Means is sensitive to initialization.

1. init='random' (Random Initialization):
   * Centroids are chosen randomly from the dataset.
2. init='k-means++' (Default in scikit-learn):
   * Chooses initial centroids far apart to avoid poor clustering.
     + The first centroid is chosen randomly.
     + Each remaining centroid is selected with a probability proportional to its squared distance from the nearest selected centroid.

A graph with a line

Description automatically generated

Understand the original dataset features 🡺 use elbow plot to determine the sharply turning point K 🡺 use chosen K to determine # of cluster to divide and group.

**Hierarchical Agglomerative Clustering (HAC)**

HAC is a type of hierarchical clustering that is **bottom-up clustering algorithm** where each data point starts as its **own cluster**, and clusters are **merged iteratively** based on closeness until a **single cluster** remains. This process creates a **binary tree** called a **dendrogram**, which represents the hierarchy of clusters.

A graph of a number of individuals

Description automatically generated with medium confidence

**Process:**

1. **Initialization**: Every point starts as its own cluster.
2. **Step A**

* Find the closest pair of clusters using a **distance metric and linkage criteria**.

1. **Step B:**

* Merge the closest clusters into one.

1. **Repeat:**

* Go back to Step A until **there is one cluster left** or until a **stopping condition is met** (like a specified number of clusters).

**Metrics to define closeness**

1. ***Distance Metric (How to Measure Distance)***

* **Euclidean Distance**:
  + Pros: Simple and commonly used.
  + Cons: Sensitive to outliers.
* **Manhattan Distance (L1 Norm):**
  + Pros: Robust to outliers.
  + Cons: Harder to optimize analytically.
* **Cosine Similarity**:
  + Measures the angle between vectors, ignoring magnitude.
  + Best For: Text data or high-dimensional data.

1. ***Linkage Criteria (How to Compare Clusters)***

* **Single Linkage (Minimum Distance):**
  + Pros: Finds elongated clusters.
  + Cons: Sensitive to noise/outliers.
* **Complete Linkage (Maximum Distance):**
  + Pros: Finds compact clusters.
  + Cons: Sensitive to outliers.

A diagram of links between two links

Description automatically generated

* **Average Linkage:**
  + Pros: Balanced between compactness and chaining effects.
* **Ward’s Method (Minimizes Variance):**
  + Merges clusters with the smallest increase in total variance.
  + Only works with Euclidean distance.
  + Works well for balanced clusters.

**Code with HAC**

1. **Model initialization**

A screenshot of a computer code

Description automatically generated

* linkage = ‘single’ 🡺 using minimum distance between 2 datapoints in 2 clusters.
* affinity = ‘euclidean’ 🡺 use Euclidean distance between 2 datapoints.
* We specify 4 clusters to form after merging 🡺 our final desired # of clusters.

1. **Generate models for all linkage**

A screenshot of a computer code

Description automatically generated

* **Use all 4-linkage method to build 4 HAC models.**
  + single: Closest points between clusters (can cause chaining).
  + average: Average distance between points in clusters (balanced).
  + complete: Farthest points between clusters (compact, sensitive to outliers).
  + ward: Minimizes variance between merged clusters (most compact).

1. **Visualize the results**

* Single Linkage

A graph with blue dots and green dots

Description automatically generated

* Average Linkage

A screen shot of a graph

Description automatically generated

* Complete Linkage

A graph with different colored dots

Description automatically generated

* Ward Linkage

A screen shot of a graph

Description automatically generated

**Evaluating Clustering Performance**

1. Within-Cluster Sum of Squared Distances (**SSD**)

* Measures compactness (similar to K-Means).
* Lower SSD = Better clusters.

1. If True Labels Are Available 🡺 Use supervised metrics:

* Homogeneity: Each cluster contains only members of one class.
* Completeness: All members of a given class are assigned to one cluster.
* V-Score: The harmonic means of homogeneity and completeness.

**Recommendation Engines**

Recommendation engines suggest items (products, services, content) to users based on their preferences, behaviors, or interactions in streaming, social media, and e-commerce.

**Types of Recommendation Engines**

1. ***Content-Based Filtering***

* Recommends items similar to **what the user has already liked based on item features**.
* Key Insight 🡺 “If you liked item A, you’ll like similar items.”
* Steps:
  + Extract features of items (e.g., genre, keywords, tags).
  + Compare items to user preferences using similarity metrics (e.g., cosine similarity).
* Example: Netflix recommends movies with similar genres or actors you’ve watched.

1. ***Collaborative Filtering***

* Recommends items based on **what similar users have liked or interacted with**.
* Key Insight 🡺 “People like you also liked these items.”
* Approaches:
  + User-User Collaborative Filtering:
    - Find users similar to the active user and recommend items they liked.
  + Item-Item Collaborative Filtering:
    - Find items similar to what the active user has rated/liked and recommend them.
* Example: Amazon’s “Customers who bought this item also bought” feature.

1. ***Hybrid / Ensemble Models***

* **Combines content-based and collaborative filtering** to improve recommendation accuracy.
* Key Insight 🡺 “Leverage the strengths of both methods.”
* Common Strategies:
  + Combine results from both models.
  + Use collaborative filtering to fill in gaps in content-based recommendations.
* Example: Netflix uses a hybrid model combining user ratings, genres, and watch history.

1. ***Recommendation as Classification***

* Treats recommendation as a classification problem where the goal is to predict whether a user will like an item.
* Approaches:
  + Use machine learning models like logistic regression, decision trees, or neural networks.
* Example: Predicting whether a user will click on an ad (CTR prediction).

**Content-based filtering with housing data**

1. Scaled the data

A screenshot of a computer

Description automatically generated

* Feature selection
  + SqFtTotLiving: total square footage of house
  + SqFtLot: lot size
  + AdjSalePrice: Adjusted sale price
* Scaling the data with StandardScaler ( )

1. Calculate pairwise distance with Euclidean distance

A screenshot of a computer

Description automatically generated 🡺

* Get pairwise distance using Euclidean Distances between every house in the dataset.
  + A smaller distance indicates 2 houses are more similar.
* A **pairwise distance matrix** shows the calculated distances between all pairs of points in a dataset.
* Each value represents the **distance** between two data points, indicating **how similar** or **different** they are.
  + The matrix has X and y axis that both indicate house 0 to house 9.
  + The diagonal is always 0 since house compared to itself is 0.
    - Distance between House 0 and House 1: 3.74 🡺 very different.
    - Distance between House 0 and House 3: 1.12 🡺 similar.
    - Distance between House 2 and House 3: 2.61 🡺 Moderately similar.

1. Query for similar houses

A screenshot of a computer code

Description automatically generated 🡺 information of House5

* If I like house5, I will want to know what other houses share the similarity with house 5.

1. Rank the similar houses to house5.

A screenshot of a computer code

Description automatically generated

**Collaborative Filtering with User Interest**

* It recommends items by finding similar users and suggesting what they liked.
* If two users have similar tastes, it assumes that preferences can be shared.

**Process**:

1. Build a User-Item Matrix that tracks interactions (e.g., likes, ratings).
2. Use Cosine Similarity to measure user similarity.
3. Find Top Similar Users to the **target user**.
4. Recommend Items liked by these similar users but not yet seen by the target user.

**Code example**

1. Construct user interest

A screen shot of a computer screen

Description automatically generated

1. Create a user-item matrix

A screenshot of a computer screen

Description automatically generated

* Extract the unique interest as matrix columns (X and Y)
  + We have 20 unique interests for each user
* Binarization: use MultiLabelBinarizer() to convert the interest into binary matrix.
  + We put 1 as user show interest to the unique interest and 0 don’t like.
  + The similarity will be constructed with rows of different users and columns of unique interest.
* User0 interest is extracted by inverse\_transform model.
  + User0 is summarized with a list where each interest is coded from binary back to original 🡺 back to original interest for user0.

1. Calculate similarity between users to user0

A close-up of words

Description automatically generated

* By using cosine similarity, we can calculate how each user share more or less similarity with one another.
* With similarity matrix (rows for users and feature for interest), we can ***calculate cosine similarity between different users based on different unique interests***.
* How is user0 similar to other uses 🡺 1st row of matrix

A screenshot of a computer

Description automatically generated

* How do other users share interest with user0 🡺 columns that share similarity between each other to user0

A screenshot of a computer

Description automatically generated

1. Find similar users and discover what exactly similarity between each users to user0

A screenshot of a computer code

Description automatically generated

* Ranks the similarity from least to most 🡺 the higher, the closer.

1. Make recommendations

A screenshot of a computer code

Description automatically generated

A screenshot of a computer

Description automatically generated

* User0’s original interest: Hadoop, Big Data, HBase, Java, Spark, Storm, Cassandra
* Top recommendation
  + MapReduce (0.57): User 9’s high similarity led to this recommendation.
  + Postgres & MongoDB (0.51): Users with similar data-tech interests influenced these.

**Imbalanced dataset**

Imbalanced classes occur when one class significantly outnumbers another in a **classification** task. It often involves with 2 major issues

1. Biased Models:

* Models tend to favor the majority class since it’s easier to optimize accuracy by predicting the majority class most of the time.

1. Misleading Accuracy:

* High accuracy can be deceptive when the model ignores the minority class.

**Ways to handle imbalanced data**

1. **Stratified Sampling**: to ensure each class is **represented equally during train-test split** 🡺 to preserve the distribution of the majority class and minority class in the data when performing data splitting.

A screenshot of a computer code

Description automatically generated

1. **Random Undersampling**

* It is used to address class imbalance by reducing the number of samples in the majority class to equalize the class distribution without changing the minority class 🡺 *to correctly classify samples from the minority class*
* **Process**
  + **Identify the Majority and Minority Classes**: Determine which class has significantly more samples.
  + **Randomly Remove Samples from the Majority Class**: Without any specific criterion, samples from the majority class are **randomly selected** and **remove**d until the class sizes are more balanced.
* **Code**

A screenshot of a computer program

Description automatically generated

* **Plot**

****

1. **Random OverSmapling**

* It creates even distribution between classes by replicating existing data points in the minority class without changing the majority class 🡺 to reduce bias toward the majority class and *correctly predict minority class samples*.
* **Process**
  + **Identify the Minority Class**
  + **Duplicate Minority Samples**: **Randomly select** samples from the minority class and **duplicate** them until the minority class size matches that of the majority class.
* **Code**

**A screenshot of a computer program

Description automatically generated**

* **Plot**

**A graph showing different colored dots

Description automatically generated**

1. **SMOTE**

* It is used to create **synthetic samples** for the minority class, rather than simply duplicating existing samples.
* By generating synthetic data, *SMOTE increases the number of samples in the minority class*, leading to a more balanced dataset without losing majority class data.
* **Process** 
  + **Apply SMOTE**:
    - For each minority class sample, find its nearest neighbors.
    - Create synthetic samples by joining between the 2 chosen samples and a randomly selected neighbor.
  + **Combine Synthetic and Original Data**: Add the synthetic samples to the original dataset, leading to an augmented dataset with a balanced class distribution.
  + **Use Balanced Dataset for Training**: Train the classifier on this new, balanced dataset, which now includes both original and synthetic samples from the minority class.
* **Code**

A screenshot of a computer

Description automatically generated

* **Plot**

A graph showing different colored dots

Description automatically generated