**Machine Learning**

Week 3

**Hierarchical Clustering**

Hierarchical Clustering is a type of unsupervised learning technique that builds a hierarchy of clusters in a **Tree-Like Structure** called **Dendrogram.**

Unlike K-Means, it does not require us to specify the number of clusters in advance.

**Types**

1. **Agglomerative (Bottom-Up Approach)**

**Process**

* Start with each data points as its own cluster
* Iteratively merge the two closest based on a distance metric (E.g. Euclidean Distance)
* Continue Merging until all the points belong to one cluster or the desired number of clusters is reached

1. **Divisive (Top-Down Approach)**

**Process**

* Start with all the data points in one cluster.
* Recursively split the cluster into smaller ones.

**Key Concepts**

* **Linkage Criteria**
  + **Single Linkage**

Minimum distance between points in two clusters.

* + **Complete Linkage**

Maximum distance between points in two clusters.

* + **Average Linkage**

Average distance between points in two clusters.

* + **Ward’s Linkage**

Minimizes the variance within clusters.

* **Distance Metric**
  + **Euclidean Distance**

It measures the **Straight-Line (As-The-Crow-Flies) distance** between two points in a Euclidean Space.

* + **Manhattan Distance**

Measures the **Sum of Absolute Differences** across dimensions.

* + **Cosine Similarity**

Measures the **Angle** between two vectors.

**Advantages**

* **No Need to Specify K**

The number of clusters is not required in advance.

* **Dendogram Visualization**

You can visually inspect the hierarchy of clusters.

* **Works Well with Small Datasets**

Implementation of agglomerative clustering is high and hence rapidly increases with larger datasets and hence it works well with small datasets.

**Disadvantages**

* **Computationally Expensive**

Hierarchical clustering can be slow on large datasets.

* **Sensitive to Noise and Outliers**

Merges Points into Clusters without the ability to undo or reassign them, potentially distorting the final structure.

* **Doesn’t Scale Well**

Computational complexity grows with the square of the number of points.

**Applications**

* **Gene Expression Analysis**

Group genes with similar expressions.

* **Customer Segmentation**

Organize customers into nested groups.

* **Document Clustering**

Group similar documents together based on topic.

**K-Means**

K-Means Clustering is an Unsupervised Machine Learning algorithm which groups unlabeled dataset into different clusters. It is used to organize data into groups based on their similarity.

It partitions the set of data points into K distinct clusters based on their features.

**How it Works?**

1. **Initialization**

Select K random points from the dataset as the initial cluster centroids.

1. **Assignment Step**

Each data point in assigned to the nearest centroid (based on distance, usually calculated by Euclidean formula).

1. **Update Step**

The centroids are recalculated as the mean of all the points assigned to the cluster.

1. **Repeat**

Steps 2 and 3 are repeated until convergence (when the centroids no longer change significantly/not at all)

**Key Concepts**

* **Centroid**

The center of a cluster, calculated as the mean of the points in that cluster.

* **Euclidean Distance**

A measure of distance between two points in a feature space.

**Formula:**



* **K**

The number of clusters you want to divide your data into. It must be specified before running the algorithm.

**Advantanges**

* **Efficiency**

K-Means is computationally efficient and works well on large datasets.

* **Scalability**

It scales well to large datasets because of its simplicity.

* **Interpretability**

The clusters formed are easy to interpret.

**Disadvantages**

* **Choosing K**

You need to specify the number of clusters (K) in advance, which might not be clear.

* **Sensitivity to Initial Centroids**

Different initializations of centroids can lead to different results.

* **Assumes Spherical Clusters**

K-Means assumes clusters are convex and spherical, which might not always be the case.

* **Sensitivity to Outliers**

Outliers can significantly affect the centroid position.

**Applications**

* **Customer Segmentation**

Grouping customers based on buying behavior.

* **Document Clustering**

Grouping similar documents in a collection.

* **Anomaly Detection**

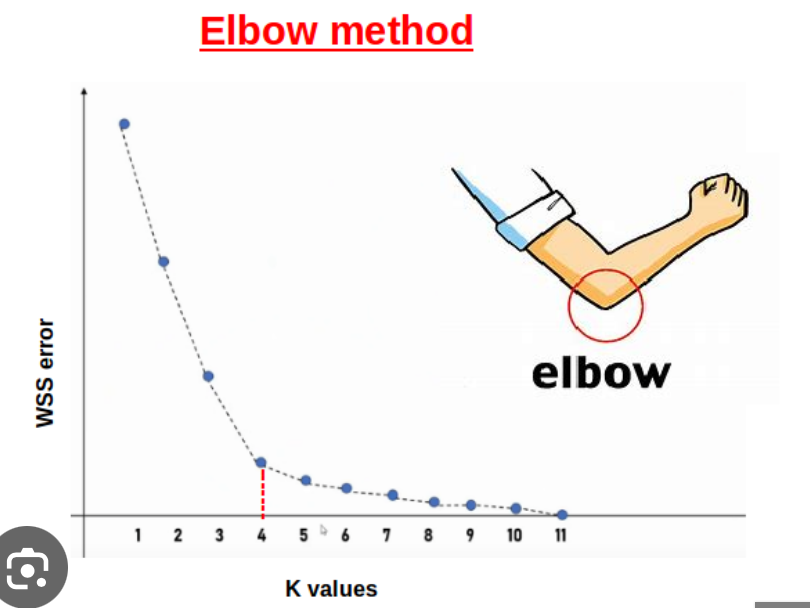
Identifying unusual data points.

**How to Choose K?**

**Elbow Method** is commonly used to determine the optimal number of clusters.

**Process**

* Run **K-Means** for a range of K Values
* Plot the within-cluster **Sum of Squares** or **Inertia**
* The **Elbow Point** on the plot in where the decrease in **Sum of Squares** slows down, indicating the optimal **K.**



**Feature Reduction**

**Feature Reduction** means reducing the dimensionality of the dataset.

**Types**

1. **Feature Selection**

Select a subset of original features based on statistical tests, importance scores, etc.

**Examples:** Mutual information, ANOVA, feature importance from RandomForest.

1. **Feature Reduction**

Create new features that summarize the original features.

**Examples:** PCA, t-SNE, Autoencoders.

**Principle Component Analysis (PCA)**

**PCA** is a **Dimensionality Reduction** technique used to reduce the number of features in a dataset while retaining as much variance (information) as possible. It helps to reduces the number of features in a dataset while keeping the most important information.

**PCA** is commonly used for data preprocessing for use with machine learning algorithms. It removes redundancy, improve computational efficiency and make data easier to visualize and analyze especially when dealing with high-dimensional data.

**Why is it Needed?**

Many datasets have redundant or correlated features. Reducing the number of features help to improve model performance, reduce overfitting, speed up training and help with visualization.

**How it Works?**

* **Standardize the Data**

Features are standardized to have mean = 0 and standard deviation = 1, because PCA is sensitive to feature scaling.

* **Compute Covariance Matrix**

Measures how features vary together. High covariance between features suggests redundancy.

* **Calculate Eigenvectors and Eigenvalues**
  + **Eigenvectors** → the **new axes** (called **principal components**).
  + **Eigenvalues** → how much **variance** each principal component carries.
* **Select Top K Principal Components**

Pick top **k** components that explain the most variance.

* **Project the Data**

Transform the original data onto these new axes (**Prinicipal Components**)

**Advantages**

* **Reduces dimensionality without much info loss**
* **Removes multicollinearity**
* **Improves algorithm efficiency**
* **Enables 2D/3D visualization of high-dimensional data**

**Disadvantages**

* **Not interpretable**
* **PCA is linear, doesn’t capture complex nonlinear relationships**
* **Sensitive to outliers**

**When to Use PCA?**

* **Too many features**
* **Features are highly correlated**
* **You want to visualize high-dimensional data**
* **You want to reduce model complexity**

**Hyperparameter Tuning**

Machine learning models have hyperparameters that you must set before training (E.g, n\_neighbours in KNN, max\_depth in DecisionTrees).

These aren’t learned from data, they control the learning process.

**Tuning**

Finding the best combination of hyperparameters that gives the highest model performance (accuracy, F1, etc.)

**Types**

1. **GridSearchCV**

Exhaustively searches every possible combination of given hyperparameters.

**How it works?**

**1.** You define a parameter grid (dictionary of all hyperparameter values you want to try).

**2.** GridSearchCV tries all combinations

**3.** For each combination, it performs a Cross-Validation to evaluate performance.

**4.** It returns the combination with the best Cross-Validation score.

**Advantages**

* Finds the **best possible** combination
* Easy to understand

**Disadvantages**

* **Slow** and **computationally expensive**, especially with many parameters

1. **RandomizedSearchCV**

RandomizedSearchCV searches a random subset of the parameter space.

It randomly selects combinations from the grid for a fixed number of iterations (controlled by n\_iter), rather than trying all of them.

**Advantages**

* Much **faster** than GridSearchCV
* Can be more **efficient** for large search spaces.

**Disadvantages**

* Might **miss the best** combination if not enough iterations
* Less exhaustive than GridSearchCV