COURSE 1

MACHINE LEARNING WITH PYTHON

INDEX

Module 1
Regression Analysis

📌 Introduction to Regression

Regression is a fundamental supervised learning technique used to model the relationship between independent variables (features) and a continuous dependent variable (target). This technique is crucial for predictive modeling, forecasting, and trend analysis across industries such as finance, healthcare, sales, and engineering.

General Regression Equation

Regression models follow a general mathematical equation:

$$\hat{y} = f(X) + \in$$

Where:

- $\hat{y}$ → Predicted value (target variable)
- X → Independent variables (predictors/features)
- $\in$ → Error term (unexplained variance)

🔹 Why is Regression Important

✔ Helps in understanding relationships between features and outcomes.

✔ Facilitates interpretability and explainability of models.

✔ Used for decision-making, forecasting, and optimization.

✔ Forms the foundation for advanced ML techniques like GLMs and deep learning regressors.

🔹 Correlation & Feature Engineering in Regression

1. Feature-Target Correlation

- Each independent variable should be strongly correlated with the target variable.
- A higher correlation means the feature has predictive power.
- **Pearson's Correlation Coefficient (r)** helps quantify this relationship:
  - r > 0.7→ Strong correlation.
  - 0.3 ≤ r ≤ 0.7→ Moderate correlation.
  - r < 0.3→ Weak or no correlation.

2. Feature-to-Feature Correlation (Multicollinearity)

- Features should **not** be highly correlated with each other.
- **Multicollinearity** can cause unstable coefficient estimates, making model interpretation difficult.

Detection:

- Correlation matrix (remove features with |r| > 0.8).
- Variance Inflation Factor (VIF) (remove features with VIF > 10).

Solutions:

✔ Remove redundant variables.

✔ Use Principal Component Analysis (PCA) or Regularization (Lasso/Ridge).

🔹 Types of Regression Models & Their Applications

1. **Simple Linear Regression (SLR)**
   - Models the relationship between **one predictor** and the target variable.
   - Formula $\hat{y} = \theta_0 + \theta_1 X$ where:
     - θ_0 → Intercept (baseline prediction)
     - θ_1 → Slope (rate of change)
     - **X** → Predictor variable

   ✔ Best for clear **linear** relationships.

   ✔ **Fails** when relationships are non-linear.

   ✔ Assumes **homoscedasticity** (constant variance in residuals).

   🟢 **Example:** Predicting **CO_2 emissions** based on engine size.

2. **Multiple Linear Regression (MLR)**
   - Extends SLR by incorporating **multiple predictors**.
   - Formula: $\hat{y} = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \cdots + \theta_n X_n$
   - Useful for **non-linear** trends.

   ✔ Risk of **overfitting** with high-degree polynomials.

   ✔ **Cross-validation and regularization** help prevent overfitting.

   🟢 **Example:** Modeling **fuel efficiency trends**.

3. **Polynomial Regression (MLR)**
    - Introduces **higher-order** terms to capture curvature.
    - Formula: $\hat{y} = \theta_0 + \theta_1 X + \theta_2 X^2 + \theta_3 X^3 + \cdots + \theta_n X^n$

    ✔ Useful for **non-linear** trends.

    ✔ Risk of **overfitting** with high-degree polynomials.

    ✔ **Cross-validation and regularization** help prevent overfitting.

    🟢 **Example:** Modeling **fuel efficiency trends**.

4. **Non-Linear Regression**

    Used when relationships cannot be captured by a straight line or polynomial function.

    - **Exponential Regression**
        - $\hat{y} = \theta_0 e^{\theta_1 X}$
        - ✔ Used for **population growth, investment returns**.
    - **Logarithmic Regression**
        - $\hat{y} = \theta_0 + \theta_1 log(X)$
        - ✔ Used for **diminishing returns, resource consumption**
    - **Sinusoidal Regression**
        - $\hat{y} = \theta_0 + \theta_1 sin(X)$
        - ✔ Used for **seasonal trends, energy demand forecasting**.

🔹 Evaluating Regression Models

To assess model performance, use the following metrics

1. **Mean Squared Error (MSE)**
    - Measures average squared error between actual and predicted values: $MSE = \frac{1}{n}\sum(y_i - \hat{y})^2$

    ✔ Lower **MSE** = **Better model fit**.

2. **Root Mean Squared Error (RMSE)**
    - Square root of MSE, easier to interpret: $RMSE = \sqrt{MSE}$

3. **Coeficient of Determination (R^2 Score)**
    - Measures how well the model explains variance in data:

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

✔ **R^2 closer to 1** = **strong model**.

✔ **R^2 near 0** = poor model.

Logistic Regression

📌 Introduction to Logistic Regression

Logistic Regression is a fundamental classification technique used in supervised learning to predict categorical outcomes. Unlike Linear Regression, which predicts continuous values, Logistic Regression transforms predictions into a probability distribution using the sigmoid function.

Key Applications:

✔ Spam detection (spam vs. not spam).

✔ Disease diagnosis (positive vs. negative test results).

✔ Loan approval (approve vs. reject).

✔ Customer churn prediction (stay vs. leave).

🔹 Logistic Regression Model

Logistic Regression applies the sigmoid (logistic) function to transform linear outputs into probabilities between 0 and 1.

Mathematical Representation:

$$P(\hat{y} = 1|x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 X_1 + \theta_2 X_2 + \cdots + \theta_n X_n)}}$$

Where:

- If p(ŷ) ≥ 0.5, classify as Class 1.
- Otherwise, classify as Class 0.
- The decision boundary is determined by adjusting the threshold.

Training a Logistic Regression Model

Training a Logistic Regression model involves optimizing parameters (θ) to minimize prediction error.

Log-Loss Function (Cross-Entropy Loss):

$$L = -\frac{1}{N}\sum[y\ log(p) + (1-y)\ log(1-p)]$$

- Penalizes incorrect confident predictions.
- The goal is to minimize log-loss to improve classification accuracy.

Optimization Techniques

There are two common optimization methods for training Logistic Regression:

1. Gradient Descent: Iteratively updates parameters to minimize log-loss.

2. Stochastic Gradient Descent (SGD): A faster variation using random subsets of data for optimization.

Multi-Class Classification with Logistic Regression

Logistic Regression can be extended for multi-class classification using:

1. ***One-vs-All (OvA):***
   - If there are K classes in total, OvA strategy will train K binary classifiers, each one distinguishing one class (positive) from all other classes (negative).
   - When making predictions, each classifier outputs a probability score.
   - The class corresponding to the classifier with the highest probability is selected as the prediction.
   - **Advantages of OvA:**
     - Simpler and computationally efficient (only K classifiers).
     - Works well when there are a moderate number of classes.
   - **Disadvantages of OvA:**
     - Can produce imbalanced classifiers (one class vs. all others).
     - Might not perform as well when classes overlap heavily.

```python
from sklearn.linear_model import LogisticRegression

model_ova = LogisticRegression(multi_class='ovr', solver='lbfgs')
model_ova.fit(X_train, y_train)
y_pred = model_ova.predict(X_test)
```

- Example: Suppose you have 3 classes:
    - Class A, Class B, Class C.

  OvA would create three separate classifiers:
    - Classifier 1: **Class A** vs. **Classes B & C**
    - Classifier 2: **Class B** vs. **Classes A & C**
    - Classifier 3: **Class C** vs. **Classes A & B**

  To predict, each classifier calculates probabilities for a data point, for example:

Classifier	Probability
A vs. All	0.35 ☑
B vs. All	0.30
C vs. All	0.25

  Since Classifier 1 has the highest probability (0.35), the model predicts Class A.

2. ***<u>One-vs-One (OvO):</u>***
    - For K classes, the OvO strategy trains K(K-1)/2 binary classifiers, each classifier distinguishing between a pair of classes.
    - When making predictions, each classifier votes for one class.
    - The class with the majority of votes across all classifiers wins.
    - **Advantages of OvO:**
        - Each classifier handles a simpler task (only distinguishing between two classes).
        - Can handle class imbalance better.
    - **Disadvantages of OvO:**
        - Computationally more expensive, especially with many classes (requires training many classifiers).
        - Can result in tie votes (solved by confidence scores or probability estimates).

```python
from sklearn.multiclass import OneVsOneClassifier
from sklearn.linear_model import LogisticRegression

model_ovo = OneVsOneClassifier(LogisticRegression(solver='lbfgs'))
model_ovo.fit(X_train, y_train)
y_pred = model_ovo.predict(X_test)
```

- Example: For the same scenario (Classes A, B, C), OvO creates 3 classifiers:
    - Classifier 1: Class A vs. Class B
    - Classifier 2: Class B vs. Class C
    - Classifier 3: Class A vs. Class C
- To predict, each classifier gives a vote, for example:

Classifier	Winner
A vs. B	A ☑
B vs. C	C ☑
A vs. C	A ☑

    - Votes: A = 2, C = 1, B = 0.
    - The predicted class is Class A, with the majority vote.

When to Use OvA vs. OvO:

Consideration	OvA Strategy	OvO Strategy
Number of classes	Moderate	Large
Computational resources available	Low (efficient)	High (expensive)
Class imbalance	Less robust	More robust
Model interpretability	Higher (fewer models)	Lower (more models)

Evaluating Logistic Regression Models

Evaluation metrics for classification tasks:

1. Accuracy

Measures the proportion of correct predictions over total predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

What it measures: Overall effectiveness of the model, but can be misleading for imbalanced datasets.

2. Precision

Measures the proportion of positive identifications that were actually correct.

$$Precision = \frac{TP}{TP + FP}$$

What it measures: How reliable your model is when it claims a positive result. High precision means fewer false alarms.

3. Recall (Sensitivity)

Measures the proportion of actual positives correctly identified by the model.

$$Recall = \frac{TP}{TP + FN}$$

What it measures: The model's ability to detect positive instances. High recall ensures fewer false negatives (missed detections).

4. F1-Score

Balances precision and recall into a single metric (harmonic mean).

$$F1 - Score = 2\ x \frac{Precision\ x\ Recall}{Precision + Recall}$$

What it measures: Overall balance between precision and recall. Useful when the dataset has uneven class distributions.

4. ROC Curve & AUC Score

- **ROC Curve:** Plots the True Positive Rate (Sensitivity) against False Positive Rate, showing the trade-off between sensitivity and specificity.
- **AUC (Area Under Curve):** A single number summarizing the ROC curve. It quantifies the model's ability to distinguish between classes (0.5 = random guessing, 1.0 = perfect discrimination).

What it measures: Model's ability to discriminate between classes at different classification thresholds.

5. Confusion Matrix

Actual \ Predicted	**Positive**	**Negative**
Positive	TP (True Positive)	FN (False Negative)
Negative	FP (False Positive)	TN (True Negative)

- **TP (True Positive):** Correctly predicted positive outcomes.
- **TN (True Negative):** Correctly predicted negative outcomes.
- **FP (False Positive):** Incorrectly predicted positive outcomes (false alarms).
- **FN (False Negative):** Missed predictions for positive outcomes.

☑ Takeaways

☑ Logistic Regression is essential for binary and multi-class classification tasks.

☑ The sigmoid function transforms linear predictions into probabilities.

☑ Log-loss is minimized to improve classification accuracy.

☑ Gradient Descent and SGD efficiently optimize logistic regression parameters.

☑ Evaluation metrics like precision, recall, and F1-score critically assess model performance.

Module 2
Classification and regression

📌 Introduction to Supervised Learning

Supervised learning involves training models on labeled data to predict outcomes for new, unseen data. The key objective is to minimize errors while ensuring generalization to new datasets, develop models that can generalize well to unseen data.

Key Learning Areas:

✔ **Classification Models** – Assign categorical labels to data points.

✔ **Regression Models** – Predict continuous numerical values.

✔ **Decision Trees and Regression Trees** – Rule-based models for decision-making.

✔ **Bias-Variance Tradeoff** – Understanding and balancing model complexity.

✔ **Ensemble Learning Techniques** – Combining multiple models for better performance.

In Module 2, we focus on:

- **Classification** → Assigning categorical labels to data points.
- **Regression** → Predicting continuous numerical values.
- **Decision Trees and Regression Trees** → Structured, rule-based learning models.
- **Understanding Tree-Based Models** → How they learn and make decisions.

📌 Classification: Categorizing Data into Groups

🔹 What is Classification?

Classification is a **supervised learning approach** where models learn to assign data points to **predefined categories** based on input features.

There are **two main types of classification problems**:

✔ **Binary Classification** → Two possible labels (e.g., spam vs. not spam).

✔ **Multiclass Classification** → More than two categories (e.g., categorizing medical conditions).

Examples of classification tasks include:

- ✔ Identifying spam emails vs. non-spam emails.
- ✔ Detecting fraudulent transactions vs. legitimate transactions.
- ✔ Categorizing customer reviews as positive, neutral, or negative.
- ✔ Diagnosing diseases based on medical records.

🔹 Multiclass Classification Strategies

Since many classification models are **binary by default**, special strategies are needed for **multiclass classification** (where there are more than two labels).

1. **One-vs-All (OvA) Strategy**
   - Also known as **One-vs-Rest (OvR)**.
   - Trains **K binary classifiers**, where each classifier learns to distinguish **one class vs. all others**.
   - The final class label is assigned based on **which classifier assigns the highest probability score**.

✔ **Simple and efficient**, especially for high-dimensional data.

✔ Works well when **one class is dominant over others**.

2. **One-vs-One (OvO) Strategy**
   - Trains **K(K-1)/2** binary classifiers, each distinguishing between **two classes at a time**.
   - A **voting mechanism** determines the final class label, based on which class wins the most pairwise comparisons.

✔ **More robust when classes are similar**, leading to **better decision boundaries**.

✔ **Computationally expensive**, especially when there are **many classes**.

✔ Use OvA when the dataset is large and computing power is limited.

✔ Use OvO when classification accuracy is a priority over computation speed.

📌 Decision Trees: Rule-Based Learning

🔹 What Are Decision Trees?

A **Decision Tree** is a **structured, rule-based model** that makes sequential decisions by **splitting the dataset** into smaller, more homogeneous groups based on feature values.

It is one of the most **intuitive and interpretable** machine learning algorithms, widely used for both **classification** and **regression** tasks.

- ✔ **Classification Trees** → Used when the target variable is categorical.
- ✔ **Regression Trees** → Used when the target variable is continuous.

🔹 How Decision Trees Work

A **Decision Tree** works by **iteratively dividing** the dataset at different feature values to **reduce uncertainty** in classification or prediction.

The structure of a **Decision Tree** consists of:

- **Root Node** → Represents the **entire dataset** and the **first decision point**.
- **Internal Nodes** → Represent **feature-based decision rules**.
- **Branches** → Show the possible **outcomes** of a decision.
- **Leaf Nodes** → Represent the **final classification (category label) or prediction (numerical value)**.

At each step, the model **selects the best feature** to split the dataset, and this process continues recursively **until a stopping criterion is met**.

🔹 Spliting Criteria for Decision Trees

Why Do We Split the Data?

The main idea behind **Decision Trees** is to repeatedly **split** the dataset in a way that **maximizes purity** in the resulting subsets.

To decide **how to split** the dataset at each node, **Decision Trees use different splitting criteria**:

1. **Entropy & Information Gain**

✔ **Entropy** measures the amount of uncertainty (or disorder) in a dataset. A **pure node** has **entropy = 0**, meaning all instances belong to the same class.
✔ **Information Gain (IG)** measures how much **entropy decreases** after a split.

✔ The feature with the **highest Information Gain** is chosen as the **best split**.

✔ **Low entropy** → The data is mostly one class (better split).

✔ **High entropy** → The data is mixed, making classification harder.

✔ **Information Gain Formula**:

$$IG = Entropy_{before} - \sum (pi\ x\ Entropy_{after})$$

pi is the proportion of instances in each resulting subset.

2. **Gini Impurity**

✔ Measures the **probability** of misclassification at a node.

✔ **Gini Impurity = 0** when the node contains a **single class** (perfect purity).

✔ The goal is to **minimize Gini Impurity** when splitting data.

✔ **Gini is computationally faster than entropy**, making it more efficient for large datasets.

✔ **Entropy is more useful for imbalanced data**, as it considers information content.

Choosing Between Gini and Entropy

- **Use Gini** when **computational speed is a priority**.
- Use **Entropy** when working with **imbalanced** class distributions.

🔹 Growing a Decision Tree: Recursive Partitioning

The **recursive partitioning** (also called **top-down induction**) process follows these steps:

1. **Start with the full dataset.**
2. **Select the best feature** to split on (based on Entropy, Gini, or other criteria).
3. **Divide the dataset** into two or more subsets.
4. **Repeat the process recursively** for each subset until a stopping criterion is met:
    - The tree reaches **maximum depth**.
    - A node has **too few samples** to split further.
    - The data in a node belongs to **a single class (pure node)**.

At the end of this process, **leaf nodes** contain either:

✔ A **class label** (for classification).

✔ A **predicted value** (for regression).

🔹 Preventing Overfitting: Pruning Decision Trees

Overfitting occurs when a Decision Tree **memorizes** training data instead of learning **generalizable patterns**.

✔ Overfitting happens when a tree is too deep, capturing noise instead of patterns.

✔ Underfitting happens when a tree is too shallow and lacks enough splits to capture relationships in the data.

To **prevent overfitting**, we use **pruning techniques**:

1. **Pre-Pruning (Early Stopping)**

   ✔ Stops tree growth early by setting a maximum depth or minimum number of samples per node.

   ✔ Limits the number of splits, preventing overly complex trees.

2. **Post-Pruning (Reduced Error Pruning)**

   ✔ The tree is **grown fully** and then pruned **after training**.

   ✔ **Branches that do not improve accuracy** are removed.

   ✔ Uses **cross-validation** to determine **which branches to prune**.

   ✔ **Pruning helps simplify trees**, making them more interpretable and improving performance on **new data**.

   ✔ **Pruning helps simplify trees**, making them more interpretable and improving performance on **new data**.

📌 Regression Trees: Predicting Continuous Values

🔹 What Are Regression Trees?

A **Regression Tree** is an extension of a **Decision Tree** used for predicting **continuous numerical values** instead of classifying categorical labels. It follows the same recursive partitioning strategy but evaluates **splitting criteria based on numerical error metrics** rather than classification purity.

✔ **Regression Trees break down a dataset into smaller and smaller subsets**, while at the same time, an associated decision tree is incrementally developed.

✔ The **final predictions** are obtained by taking the **average of values** in the leaf nodes rather than majority voting (as in classification trees).

Regression Trees are **widely used** for:

✔ **Sales forecasting** (e.g., predicting product demand).
✔ **Real estate pricing** (e.g., estimating house prices based on location and features).
✔ **Medical prognosis** (e.g., predicting hospital stay durations based on patient data).

🔹 How Regression Trees Work

The **main goal** of a Regression Tree is to split the data in a way that **minimizes prediction error**.

Unlike classification trees, which maximize class purity using **Entropy or Gini Impurity**, Regression Trees evaluate splits using **variance reduction methods**.

Step-by-Step Process

1. **Start with the entire dataset** as the root node.
2. **Identify the best feature and split point** that minimizes prediction error.
   When selecting split points, feature values are sorted, for continuous features, potential splits are **midpoints between consecutive unique values.**
3. **Recursively partition the data** into smaller subsets until a stopping criterion is met:
   - The tree reaches **maximum depth**.
   - The number of data points in a node **falls below a threshold**.
   - Further splits **do not improve prediction accuracy significantly**.
4. **Make predictions at the leaf nodes** → Each leaf node contains the **average value of the target variable**.

At the end of this process, a Regression Tree produces a **piecewise constant function**, meaning the dataset is split into **intervals**, and the model predicts a **constant value** (the mean of training samples) for each interval.

🔹 Spliting Criteria for Regression Trees

Since Regression Trees predict **continuous values**, they use different **split evaluation criteria** compared to Classification Trees.

1. Mean Squared Error (MSE) Reduction

✔ The most **common** method for deciding where to split the data in a Regression Tree.

✔ **MSE measures the variance of the target values within a node**. The lower the MSE, the **better the split**.

✔ **Formula for MSE**:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

- y_i -> Actual target values in the node.
- $\hat{y}$ -> Mean of the target values in the node.
- n -> Number of samples in the node.

✔ At each node, the feature and split value that result in the lowest weighted MSE is chosen.

2. Mean Absolute Error (MAE) Reduction

✔ Similar to MSE but less sensitive to outliers.

✔ Uses the absolute difference instead of squared errors.

✔ Formula for MAE:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}|$$

✔ MAE is useful when data contains extreme outliers, as it does not penalize them as heavily as MSE.

3. **Variance Reduction**

✔ The goal of Regression Trees is to reduce variance in the dataset.

✔ Variance measures how spread out the target values are in a given node.

✔ The split that results in the greatest variance reduction is selected.

✔ Formula for Variance Reduction:

$$Variance = \frac{1}{n}\sum(y_i - \bar{y})^2$$

- Where $\bar{y}$ is the mean of the target values in the node.

✔ Lower variance = **better homogeneity** in target values within a node = **better prediction accuracy**.

📌Comparing Decision Trees - Classification vs. Regression

Feature	Classification Trees	Regression Trees
Prediction Output	Discrete Class Labels (e.g., "Dog" or "Cat")	Continuous Numeric Values (e.g., price, temperature)
Splitting Criteria	Entropy, Gini Impurity	Mean Squared Error (MSE), Mean Absolute Error (MAE)
Use Cases	Email Spam Detection, Disease Diagnosis	Sales Prediction, Stock Price Forecasting
Final Decision at Leaf Node	Email Spam Detection, Disease Diagnosis	Average Value of Data Points

✔ **Classification Trees assign categories**, while **Regression Trees predict continuous values**.

✔ **Both trees use recursive splitting**, but **their criteria differ** based on whether the target is **categorical or numerical**.

Other Supervised Learning Models

📌 Introduction to Supervised Learning

Supervised learning involves training models on labeled data to predict outcomes for new, unseen data. The key objective is to minimize errors while ensuring generalization to new datasets, develop models that can generalize well to unseen data.

In Module 3, we focus on:

- **K-Nearest Neighbors (KNN)** – An instance-based algorithm that classifies data based on similarity.
- **Support Vector Machines (SVMs)** – A powerful classification algorithm that finds the optimal decision boundary.
- **Bias-Variance Tradeoff** – Understanding and balancing model complexity.
- **Ensemble Learning (Bagging & Boosting)** – Improving model performance by combining multiple classifiers.

📌 Support Vector Machines (SVM)

🔹 What is Support Vector Machines (SVM)?

Support Vector Machines (SVMs) are supervised learning algorithms used for both classification and regression, with a primary focus on finding the optimal decision boundary that best separates different classes in a dataset.

✔ SVM is powerful for **high-dimensional** spaces where clear class separation is required.

✔ It **works well for both linearly and non-linearly separable data** by mapping data into higher dimensions using kernel functions.

✔ It aims to maximize the margin between data points belonging to different classes.

Unlike traditional classifiers like KNN or Logistic Regression, SVM does not rely on probability scores but instead finds the most optimal hyperplane that distinctly classifies data points.

Use SVM when:

✔ Use SVM for small- to medium-sized datasets with complex, non-linear boundaries.

✔ **Use SVM when interpretability is not a major concern** but accuracy is the priority.

✔ Avoid SVM for extremely large datasets due to high computational costs.

🔹 How SVM works

SVM classifies data by finding the best possible hyperplane that separates different classes while maximizing the margin (distance between the closest data points of different classes, known as support vectors).

Key Concept: ***Support Vectors***

✔ Support Vectors are the closest data points to the hyperplane.

✔ These points define the margin and influence the decision boundary.

✔ The model does not consider other data points when forming the boundary, making SVM highly robust to noise.

1. Identifying the Optimal Hyperplane

- In two-dimensional space, the hyperplane is simply a line that separates data points.
- In three-dimensional space, the hyperplane is a plane that divides the dataset.
- In higher-dimensional spaces, the hyperplane is a mathematical decision boundary.

2. Hard Margin vs Soft Margin

✔ Hard Margin SVM (Strict separation)

- The model assumes that all data points can be perfectly separated by a linear boundary.
- Works well for **clean**, noise-free datasets.
- **Limitations** → If classes overlap even slightly, the model fails completely

✔ Soft Margin SVM (Tolerates misclassifications)

- Introduces a flexibility parameter (C) that allows some misclassification to balance generalization.
- The lower the C parameter, the more misclassification is allowed (higher bias, lower variance).
- The higher the C, the stricter the boundary (lower bias, higher variance).

✔ In real-world applications, Soft Margin SVM is preferred as real-world data is rarely perfectly separable.

3. Kernel Trick: Handling Non-Linearly Separable Data

In many datasets, a simple linear boundary cannot separate classes. To solve this, SVM uses kernel functions to project data into higher dimensions, making it easier to separate.

Common Kernels in SVM

✔ **Linear Kernel:**

- Works well if data is linearly separable (can be separated by a straight line).
- Fast and efficient for simple problems**.**
- The decision function is simply:

$$f(x) = w.x + b$$

✔ **Polynomial Kernel:**

- Suitable when **curved decision boundaries** are required.
- Captures complex interactions between features.
- The Kernel function:

$$K(x_i, x_j) = (x_i.x_j + c)^d$$

✔ **Radial Basis Function (RBF) Kernel:**

- Maps data into **infinite-dimensional space**, making even complex datasets separable.
- The most commonly used kernel for handling non-linearly separable data.
- The function:

$$K(x_i, x_j) = e^{-\gamma\left||x_i - x_j|\right|^2}$$

✔ **Sigmoid Kernel:**

- Mimics neural networks and is used for **binary classification problems**.
- The function:

$$K(x_i, x_j) = tanh\ (\alpha x_i . x_j + c)$$

✔ **Choosing the right Kernel:**

- Use **Linear Kernel** if the dataset is linearly separable.
- Use RBF Kernel for complex, high-dimensional problems.
- Use Polynomial Kernel if interactions between features matter.

4. Kernel Trick: Handling Non-Linearly Separable Data

To improve SVM performance, two **main hyperparameters** need tuning:

✔ **C (Regularization Parameter):**

- Controls the tradeoff between bias and variance.
- Higher C → Stricter separation, lower bias, but more prone to overfitting.
- Lower C → More flexible boundary, higher bias, but better generalization.

✔ **Gamma (γ Parameter for RBF Kernel):**

- Controls the influence of each training example.
- Higher γ → Model becomes too sensitive to individual points, leading to overfitting.
- Lower γ → SVM fails to capture patterns properly.

✔ **Using Grid Search for Hyperparameter Tuning:**

- Hyperparameters C and gamma should be fine-tuned using cross-validation to find the best combination.

📌 K-Nearest Neighbors (KNN)

🔹 What is K-Nearest Neighbors (KNN)?

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for both classification and regression. Unlike other machine learning algorithms that explicitly learn a model during training, KNN memorizes the dataset and makes predictions based on similarity.

KNN assumes that data points close to each other share similar properties and that classification can be determined based on proximity to labeled examples.

✔ It is a **lazy learning algorithm**, meaning no actual training phase occurs—instead, computations happen at the time of prediction.

✔ It is **simple and effective**, working well with small datasets and low-dimensional feature spaces.

✔ It **does not assume any distribution** about the data, making it useful for non-linear decision boundaries.

Use KNN when:

✔ Use KNN when the dataset is small and interpretability is needed.

✔ Avoid KNN for large datasets due to slow prediction speed.

🔹 How KNN Works

The core principle of KNN is that it assigns a label (classification) or predicts a value (regression) for a new data point based on its K closest neighbors in the training set.

The process involves four key steps:

1. Choose a value for K (the number of nearest neighbors to consider).
2. Calculate the distance between the new data point and all training data points.
3. Find the K-nearest neighbors (smallest distances).
4. Make a prediction:
    - For classification → Assign the most frequent class among the K neighbors (majority vote).
    - For regression → Take the average (or weighted average) of the target values of the K neighbors.

✔ The performance of KNN depends on choosing an optimal K value and an appropriate distance metric.

🔹 Choosing the Right K Value

Selecting the optimal K value is a crucial step in achieving a well-performing KNN model. The choice of K impacts the bias-variance tradeoff:

- Small K (e.g., K=1, K=3) → Leads to Overfitting

  ✔The model becomes too sensitive to noise and classifies based on just one or two points.

  ✔High variance and low bias (too flexible).

- Large K (e.g., K=20, K=50) → Leads to Underfitting

  ✔The model becomes too general and fails to capture local structures in data.

  ✔Low variance and high bias (too rigid).

- Optimal K → Usually found via cross-validation.

🔹 Finding the Optimal K Using Accuracy Analysis

We determined the best value of K by training multiple KNN models with different values of K and evaluating their accuracy on a test set.

The process followed these steps:

1. **Define a range of K values to test**
   - We started with a range of small to large values of K (e.g., from 1 to 50).
   - The goal was to observe how accuracy changes with increasing K.
2. **Train multiple KNN models**
   - A separate KNN model was trained for each selected K value.
   - Each model made predictions on the test set and the accuracy was recorded.
3. **Plot accuracy vs. K values**
   - The accuracy scores were plotted against the corresponding K values.
   - This helped visualize how accuracy changes with different K values.

4. **Identify the best K**
    - The optimal K was chosen as the one that maximized accuracy while maintaining stability.
    - If accuracy fluctuated significantly, we selected a slightly larger K for smoother performance.

Interpreting the Results:

- If the accuracy is highest at very low K values (K=1, K=3), the model is likely overfitting to the training data.
- If the accuracy remains stable for mid-range K values (K=5 to K=15), these values are usually good choices.
- If the accuracy declines for large K values (K > 30), the model is underfitting, failing to capture important decision boundaries.

✔ The best practice is to select the K value that provides the highest accuracy while preventing excessive fluctuations due to noise.

🔹 Distance Metrics in KNN

Since KNN relies on measuring distances, selecting the right distance metric is critical for performance.

1. **Euclidean Distance (Most Common)**

    The default metric, representing the shortest straight-line distance between two points:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

    ✔ Works well with continuous numerical data.

    ✔ Assumes all features are equally important.

2. **Manhattan Distance (City Block Distance)**

    The sum of absolute differences between features values

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

    ✔ More robust for high-dimensional data.

    ✔ Works well when features have different units of measurement.

3. **Minkowski Distance (Generalized Distance Metric)**

   A generalization of Euclidean and Manhattan distances:

$$d = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

   - When **p = 1**, Minkowski reduces to **Manhattan Distance**.
   - When **p = 2**, Minkowski reduces to **Euclidean Distance**.

✔ Allows flexibility in distance calculations.

🔹 Handling Imbalanced Classes in KNN

KNN can be biased towards the majority class if the dataset is imbalanced.

To address this, we can:

✔ **Use Weighted KNN** → Assign higher weights to closer neighbors, reducing the influence of distant points.

✔ **Use Data Resampling** → Oversample the minority class or undersample the majority class.

Weighted KNN formula:

$$y = \frac{\sum_{i=1}^{K} w_i y_i}{\sum_{i=1}^{K} w_i}$$

where:

- w_i is the inverse distance weight $\frac{1}{d_i}$.

✔ This approach ensures that closer neighbors contribute more to predictions.

🔹 Handling Feature Scaling in KNN

Since distance-based algorithms are sensitive to feature magnitudes, feature scaling is essential.

✔ Min-Max Scaling → Scales data to the [0,1] range.

✔ Standardization (Z-score Normalization) → Centers data with mean μ=0 and standard deviation σ = 1.

✔ Feature scaling ensures that all features contribute equally to distance calculations.

📌 Bias-Variance Tradeoff.

🔹 What is the Bias-Variance Tradeoff?

The **Bias-Variance Tradeoff** is a **fundamental concept in machine learning** that describes the **tension between model complexity and generalization**.

The goal is to develop a model that **performs well on both training and unseen test data**, meaning it **neither underfits nor overfits**.

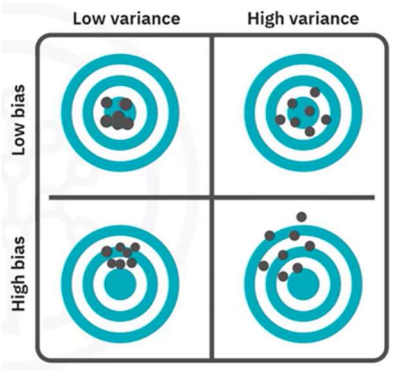
When a model has high bias, it oversimplifies the data and fails to learn meaningful patterns.

When a model has high variance, it memorizes the training data instead of generalizing patterns.

- ✔ **Bias** measures how well the model represents the true relationship between inputs and outputs.
- ✔ **Variance** measures how sensitive the model is to variations in training data.

A model with **high bias** tends to underfit the data, missing important patterns.

A model with **high variance** tends to overfit the data, capturing noise instead of actual patterns.



Think of bias as how on-target or off-target the darts are.

Variance measures how spread out the darts are, representing precision.

The top two boards demonstrate low bias, meaning they are more accurate, while the bottom two show higher bias, making them less accurate. Think of bias as how on-target or off-target the darts are.

The dart boards on the right display higher variance, meaning the darts are more spread out, while the boards on the left show lower variance, with the darts grouped closer together.

As shown on the top left board, achieving a high score requires both low bias for accuracy and low variance for precision.

What is Bias?

Bias represents systematic errors introduced when a model makes strong assumptions about the data's structure. It is the measure of how far off the model's predictions are from the true values.

✔ **High Bias (Underfitting)**

- The model is too simple and fails to capture the underlying pattern in the data.
- It assumes a rigid structure (e.g., trying to fit a straight line to a highly curved dataset).
- Leads to poor accuracy on both training and test data.

✔ **Characteristics of High-Bias Models**

- Fails to learn from training data.
- High training error and high-test error.
- Predictions do not change much when training on different datasets.

✔ **Examples of High Bias Models**

- Linear Regression on non-linear data.
- A decision tree with only one or two splits.
- A low-degree polynomial regression model trying to fit complex data.

What is Variance?

Variance refers to how much a model's predictions change when trained on different subsets of the data. A high-variance model is too complex and captures noise instead of the true pattern.

✔ High Variance (Overfitting)

- The model memorizes training data, making it highly sensitive to small variations.
- Works extremely well on training data but fails to generalize to unseen data.
- Any small fluctuations in the dataset can drastically change predictions.

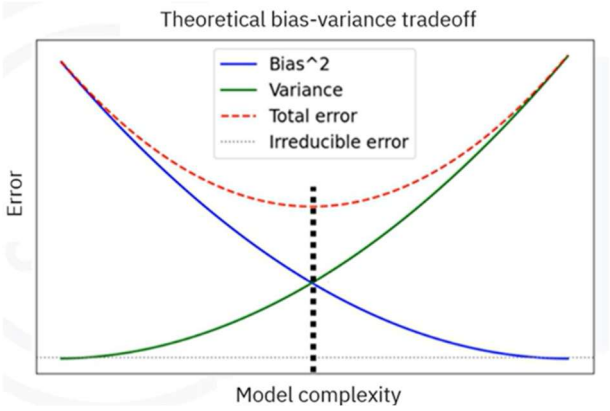
✔ Characteristics of High-Variance Models

- Very low training error but high test error.
- Highly sensitive to small changes in the dataset.
- Predictions change drastically with different training samples.

✔ Examples of High-Variance Models

- A deep decision tree without pruning.
- KNN with K=1 (memorizing every training point).
- A high-degree polynomial regression model fitting every data point exactly.

🔹 Balancing Bias and Variance



This plot illustrates how bias and variance changes as the model becomes more complex and better at predicting the data it's trained on.

As model complexity increases, bias, represented by the blue curve, tends to decline, while variance, shown by the green curve, rises.

When model complexity is low, bias is high, leading to poor predictions even on training data. This is known as underfitting.

Conversely, high model complexity results in high variance, meaning the model becomes overly sensitive to the training data and performs poorly on unseen data, resulting in overfitting.

However, there's a crossover point marked by the vertical dashed line where the model's complexity is just right.

There will always be some generalization error that cannot be eliminated, such as random noise in the data.

A model must find balance between **bias and variance** to **generalize well** on new data.

✔ A good model is one that finds the right balance between bias and variance.

Model Complexity	Bias	Variance	Training Error	Test Error
Simple Model (Underfitting)	🔴 High	☑ Low	❌ High	❌ High
Balanced Model (Good Fit)	☑ Low	☑ Low	☑ Low	☑ Low
Overly Complex Model (Overfitting)	☑ Low	🔴 High	☑ Very Low	❌ High

📌 Bias-Variance and Ensemble Learning

🔹 What is Ensemble Learning?

Ensemble Learning is a **machine learning technique** that **combines multiple models** to produce a **stronger, more robust predictive model**. Instead of relying on a **single classifier**, ensemble learning aggregates predictions from multiple weak models, improving **generalization, accuracy, and stability**.

✔ **It mitigates the bias-variance tradeoff** by balancing model complexity.

✔ **It reduces overfitting (high variance) and underfitting (high bias).**

✔ **It is particularly useful for improving weak learners** such as decision trees.

🔹 How Bias and Variance Relate to Ensemble Learning

Bias and variance contribute **differently** to model performance. **Ensemble methods help control them** by combining multiple models.

Ensemble Method	Reduces Bias?	Reduces Variance?	How It Works
Bagging (Bootstrap Aggregating)	❌ No	☑ Yes	Averages multiple independent models to reduce overfitting.
Boosting	☑ Yes	❌ No	Sequentially trains weak models to improve prediction accuracy.

✔ **Bagging is useful when variance is high (to stabilize predictions).**

✔ **Boosting is useful when bias is high (to improve weak models).**

🔹 Reducing Variance - Bagging

What is bagging?

✔ Bagging (Bootstrap Aggregating) is an ensemble technique that reduces variance by training multiple independent models on different random subsets of the data.

✔ It ensures that each individual model learns slightly different aspects of the dataset, reducing overfitting.

✔ The final prediction is made by aggregating predictions:

- Classification → Majority voting (the most common class among models is chosen).
- Regression → Averaging predictions from all models.

How Bagging Works

1. Create multiple bootstrap samples → Randomly select subsets of training data with replacement (some samples may appear more than once).
2. Train multiple independent models on different subsets.
3. Each model makes a prediction, and the final result is obtained by majority voting (classification) or averaging (regression).

✔ Since models are trained on different subsets of data, their errors are less correlated, reducing overall variance.

Why does Bagging work?

- It smooths out fluctuations by training on diverse data samples.
- Works best with high-variance models (e.g., deep decision trees).
- Not very useful for high-bias models (like logistic regression), because adding weak models does not fix fundamental underfitting.

🔹 Reducing Bias - Boosting

What is Boosting?

✔ Boosting is an ensemble technique that reduces bias by sequentially training models, where each new model focuses on correcting errors made by previous models.

✔ Unlike Bagging, which trains models independently, Boosting builds models sequentially, making them more dependent on previous iterations.

✔ The final prediction is a weighted sum of all weak models.

How Boosting Works

1. **Train an initial weak model (e.g., a shallow decision tree).**
2. **Identify misclassified or high-error data points** from the previous model.
3. **Assign higher weights** to these "hard-to-classify" points so that the next model focuses on them.
4. **Train a new model on the updated dataset** with adjusted weights.

5. **Repeat for multiple iterations**, creating a sequence of progressively better models.
6. **Combine the predictions** from all weak models using a weighted sum.

✔ **Boosting turns weak learners into a strong ensemble by improving prediction step by step.**

🔹 Comparing Bagging and Boosting

✔ **Use Bagging when variance is high** (to stabilize predictions).
✔ **Use Boosting when bias is high** (to improve weak models).
✔ **Both methods improve generalization and performance**.

Method	Objective	How It Works	Best Use Case
Bagging	Reduce variance	Train multiple independent models on different random subsets of data	When variance is high (e.g., Random Forests)
Boosting	Reduce bias	Train models sequentially, each correcting the previous one	When bias is high (e.g., XGBoost, AdaBoost)

📌 Algorithms strengths and weaknesses

Algorithm	Strengths	Weaknesses
KNN	✔ Simple, non-parametric, works well with small data.	✘ Computationally expensive for large datasets.
Decision Tree	✔ Fast inference, interpretable rules	✘ Prone to overfitting without pruning.
SVM	✔ Effective for high-dimensional spaces.	✘ Sensitive to noise, requires careful tuning.
Logistic Reg.	✔ Works well for linearly separable data.	✘ Struggles with complex decision boundaries.

Module 3
Clustering

📌 Clustering Strategies and Real-World Applications

🔹 What is Clustering?

Clustering is an **unsupervised machine learning technique** that groups **similar data points** into clusters based on their **features and relationships**.

It's a **pattern discovery technique** that finds **hidden structures** in datasets.

It assigns **similar data points to groups (clusters) based on feature similarity**.

✔ Unlike classification, clustering does not require **labeled data**; instead, it **identifies patterns within the dataset** to form natural groupings.

✔ **Clusters are formed naturally** based on proximity in an **N-dimensional feature space**.

✔ It is used in applications where patterns exist but predefined labels are unavailable.

Real-World Applications of Clustering.

- **Customer Segmentation** → Grouping customers by **shopping habits** to personalize marketing.
- **Image Segmentation** → Identifying regions in medical imaging (e.g., tumor detection).
- **Anomaly Detection** → Detecting **fraudulent transactions** or **equipment failures**.
- **Genetic Data Analysis** → Grouping similar DNA sequences to study genetic relationships.
- **Data Summarization** → Reducing large datasets by summarizing them into **representative clusters**.

🔹 Clustering uses

🚀 Overall, clustering is used to simplify data, reveal hidden relationships, and enhance decision-making across industries.

It's a powerful tool in data analysis because it reveals patterns and relationships that may not be apparent in raw data, it is widely used for:

✔ In **exploratory data analysis**, clustering **uncovers natural groupings**, such as customer segmentation, for targeted marketing.

✔ Boosts **pattern recognition** by grouping similar objects and aiding in image segmentation, such as detecting medical abnormalities.

✔ Clustering helps **anomaly detection** by identifying outliers and detecting fraud or equipment malfunctions.

✔ In **feature engineering**, clustering creates new features or reduces dimensionality, improving model performance and interpretability.

✔ In **data summarization**, clustering simplifies data by summarizing it into a small number of representative clusters.

✔ Clustering **reduces data size** by replacing data points with cluster centers, which is useful for image compression.

📌 Types of Clustering Methods

Clustering methods vary in their approach to grouping data points. The three main categories are:

Type	How It Works	Best Use Case
Partition-Based	Divides data into **K clusters** (fixed number).	Large, well-separated datasets.
Density-Based	Groups **high-density regions** and ignores noise.	Data with irregular patterns.
Hierarchical	Creates a **tree of nested clusters**.	Small to mid-sized datasets with hierarchical relationships.

Support Vector Machines (SVMs) are supervised learning algorithms used for both classification and regression, with a primary focus on finding the optimal decision boundary that best separates different classes in a dataset.

✔ SVM is powerful for **high-dimensional** spaces where clear class separation is required.

✔ It **works well for both linearly and non-linearly separable data** by mapping data into higher dimensions using kernel functions.

✔ It aims to maximize the margin between data points belonging to different classes.

Unlike traditional classifiers like KNN or Logistic Regression, SVM does not rely on probability scores but instead finds the most optimal hyperplane that distinctly classifies data points.

1.Partition-Based Clustering (K-Means)

Partition-based clustering divides the dataset into **non-overlapping groups**, where **each data point belongs to only one cluster**.

✔ **K-Means** is the most commonly used **partition-based clustering algorithm**.
✔ It **minimizes intra-cluster variance** by iteratively refining cluster assignments.

Example:

A retail company applies K-Means clustering to segment customers into three groups based on purchasing behavior.

🚀 **Strengths**: Efficient, scalable.

⚠️ **Limitations**: Requires pre-defining **K (number of clusters)**.

2. Density-Based Clustering (DBSCAN, HDBSCAN)

Density-based clustering identifies high-density regions and groups points that are closely packed together.

✔ Unlike K-Means, it does not require specifying K (number of clusters).

✔ **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) finds arbitrarily shaped clusters.

✔ **HDBSCAN** (Hierarchical DBSCAN) improves upon DBSCAN by adjusting density thresholds dynamically.

Example:

Detecting fraudulent transactions → Unusual spending patterns (outliers) form separate clusters.

🚀 **Strengths:** Handles noisy data and irregular cluster shapes.

⚠️ **Limitations:** Struggles with varying density clusters.

3. Hierarchical Clustering (Agglomerative & Divisive)

Hierarchical clustering creates a tree of clusters called a dendrogram, showing the relationships between clusters.

🚀 **Strengths**: Provides interpretable cluster relationships.

⚠️ **Limitations**: Computationally expensive for **large datasets**.

✔ **Agglomerative Clustering (Bottom-Up Approach)**

- Each data point starts as an individual cluster.
- Clusters merge iteratively based on similarity.
- The process continues until a predefined number of clusters is reached.

Steps in Agglomerative Clustering:

1. Compute pairwise distances between data points.
2. Form a distance matrix and merge the closest clusters.
3. Update the distance matrix with new cluster distances.
4. Repeat until the required number of clusters is reached.

✔ **Divisive Clustering (Top-Down Approach)**

- The entire dataset starts as one large cluster.
- Clusters are split recursively until a stopping condition is met.

Steps in Divisive Clustering:

1. Start with one cluster containing all data points.
2. Identify the most dissimilar points and split the cluster.
3. Repeat recursively, continuing to split clusters.
4. Stop when clusters meet a predefined threshold.

✔ **More computationally expensive** than Agglomerative clustering.

✔ **Distance Metrics for Clustering:**

Hierarchical clustering needs two thing in order for be trained:

- A way to measure distance, in order determine the proximity of data points.
- A criterion to merge clusters, this criterion is names **linkage**, and in simple words, is how we obtain the distance to measure the proximity between data points. The distance metrics are:
    - **Single Linkage** → Uses the **closest** data points in two clusters to measure the distance.

- **Complete Linkage** → Uses the **farthest** points in two clusters.
- **Average Linkage** → Uses the **mean** distance between clusters.

📌 K-Means

K-Means Clustering is a **centroid-based, unsupervised learning algorithm** that partitions a dataset into **K distinct clusters** based on feature similarity. Unlike **supervised learning**, where models learn from labeled data, K-Means identifies **inherent structures** in unlabeled datasets, grouping similar data points together.

🔹 Understanding K-Means Clustering

K-Means **divides data into K non-overlapping clusters**, aiming to **minimize intra-cluster variance** while ensuring **maximum dissimilarity between clusters**. Each cluster is represented by a **centroid**, which is iteratively updated to refine cluster assignments.

✔ **Key concepts:**

- **Centroid** → The **mean position** of all points in a cluster.
- **Cluster Assignment** → Each data point is assigned to the **nearest centroid** based on distance.
- **K Value** → The number of clusters, **must be defined before running the algorithm**.

🔹 K-Means Algorithm: Step-by-Step

The K-Means algorithm follows an **iterative process** to optimize cluster assignments.

1. Initialize Centroids
   - Choose the **number of clusters (K)**.
   - Randomly **select K initial centroids** from the dataset.
2. Assign Data Points to Nearest Centroids
   - Compute **distances from each data point to all centroids** (commonly using **Euclidean distance**).
   - Assign each point to the **nearest centroid**.
3. Update Centroids
   - Recalculate **the centroid of each cluster** as the mean of all assigned points.

4. Repeat Until Convergence
    - Reassign points to the new nearest centroids.
    - Continue until centroids no longer move or a maximum number of iterations is reached.

✔ The algorithm converges when centroid positions stabilize or maximum iterations are reached.

🔹 Distance Metrics Used Clustering

K-Means Clustering relies on a **distance metric** to determine **which data points belong to which cluster**. The most commonly used metric is **Euclidean distance**, but **other distance measures** can also be used depending on the dataset and its characteristics.

✔ Euclidean Distance (Most Common in K-Means)

Simple and computationally efficient.

Works well when **data is normalized and clusters are spherical**.

Directly measures how close a point is to its centroid.

⚠️ **Limitations:**

- Assumes all features contribute **equally** to distance calculations.
- **Sensitive to feature scaling** (features with large values dominate).

Solution: Standardize features (e.g., Z-score normalization) before applying K-Means.

Formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Where:

- *x* and *y* are two points in **n-dimensional space**.
- x_i and y_i are their respective coordinates.
- The sum calculates **squared differences for each feature**, and the square root gives the final distance.

✔ Manhattan Distance (L1 Norm)

- Measures distance as the sum of absolute differences between feature values.
- Preferred when features are highly independent (e.g., city block/grid data).

⚠ Why K-Means Doesn't Use It Often?

- Creates **diamond-shaped clusters** instead of circular ones.
- Less sensitive to outliers but **not ideal for centroid-based algorithms**.

When to use -> If data is high-dimensional but sparse (e.g., text data with word counts).

Formula:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

✔ Cosine Distance (Similarity Measure)

Used for **high-dimensional data** (e.g., text, embeddings, gene expressions).

Measures how "aligned" two vectors are rather than their absolute difference.

Often used in document clustering (e.g., NLP applications).

⚠ Why K-Means Doesn't Use by default?

- Requires **vector normalization** before use.

When to use -> If feature values represent **angles/directions** (e.g., word frequency vectors).

Formula (Cosine similarity):

$$\cos(\theta) = \frac{x \,.\, y}{||x||\ ||y||}$$

Where:

- $x \,.\, y$ is the **dot product** of two vectors.
- $||x||$ and $||y||$ are the magnitudes (lengths) of the vectors.

✔ Mahalanobis Distance (Accounts for Feature Correlation)

Accounts for correlations between features (unlike Euclidean distance).

Creates **elliptical** rather than circular clusters, making it useful when features are dependent.

⚠️ Why K-Means Doesn't Use It Often?

- Requires computing the **covariance matrix**, which is expensive for large datasets.
- Works **poorly with small sample sizes** or highly imbalanced data.

When to use -> If **features are correlated** (e.g., financial data, chemical compositions).

Formula:

$$\cos(\theta) = \sqrt{(x-y)^T S^{-1} (x-y)}$$

Where:

- S^{-1} is the inverse covariance matrix, adjusting for feature correlations.

✔ Choosing the Right Distance Metric

🚀 Key Takeaways:

- **Euclidean Distance** is the default for K-Means because it is computationally simple and aligns well with centroid-based clustering.
- **Manhattan and Cosine Distances** are better for non-Euclidean feature spaces, but they are **not ideal for centroid-based clustering**.
- **Mahalanobis Distance** is powerful for correlated features, but **computationally expensive**.

Distance Metric	Best for...
Euclidean Distance	Well-separated, spherical clusters
Manhattan Distance	Grid-like/sparse data (e.g., city distances)
Cosine Distance	Text or high-dimensional sparse data
Mahalanobis Distance	Correlated features, elliptical clusters

🔹 Challenges and Limitations of K-Means

While K-Means is efficient, it **relies on specific assumptions** that may not hold in all datasets.

✔ **Clusters are Convex and Isotropic**

- K-Means assumes clusters are **spherical** and **well-separated**.
- If clusters have **irregular shapes**, K-Means struggles to separate them properly.

✔ **Equal Cluster Sizes**

- The algorithm assumes **clusters contain a similar number of points**.
- When clusters are **imbalanced**, K-Means may assign more points to the larger cluster, **distorting results**.

✔ **Sensitivity to Initial Centroids**

- Since centroids are **randomly initialized**, different runs may produce **different cluster assignments**.
- **Solution** → The **K-Means++ initialization method** reduces this issue by choosing centroids **far apart** initially.

✔ **Sensitivity to Noise and Outliers**

- K-Means uses **variance as a criterion**, making it sensitive to **noisy data**.
- **Solution →** Preprocess data by **removing outliers** or **scaling features**.

🔹 How to Choose the Optimal Number of Clusters (K)?

One major challenge in K-Means is determining the **best K value**. Several techniques help identify the most suitable number of clusters.

1. Elbow Method

The **Elbow Method** plots **inertia (within-cluster sum of squares, WCSS)** as a function of K.

The goal is to **find the "elbow point"**, where adding more clusters **yields diminishing returns**.

✔ **Steps to Use the Elbow Method:**

1. Compute K-Means for **multiple values of K** (e.g., K=1 to K=10).
2. Record the **inertia (sum of squared distances between points and their centroids)**.
3. Plot the **K values vs. inertia** and **identify the "elbow"**, where inertia **stabilizes**.

✔ **Interpretation:**

- If **K is too small** → Clusters **contain highly dissimilar points**, increasing WCSS.
- If **K is too large** → Clusters become **too small**, increasing variance unnecessarily.
- The **best K is where the curve bends** (elbow point).

2. Silhouette Score

Measures how well-separated clusters are by comparing:

- **Cohesion** (how close a point is to others in clusters).
- **Separation** (how far it is from points in the nearest cluster).

✔ **Formula:**

$$S = \frac{b - a}{\max(a, b)}$$

Where:

- **a** = Mean intra-cluster distance (cohesion).
- **b** = Mean nearest-cluster distance (separation).

✔ **Ranges from -1 to 1:**

- **Higher scores (close to 1)** → Well-separated clusters.
- **Low scores (near 0)** → Overlapping clusters.
- **Negative scores** → Points assigned to the wrong cluster.

3. Davies-Bouldin Index

Measures the **average similarity ratio between each cluster and its most similar cluster.**

Lower values indicate **better clustering performance**

✔ Used in cases where the Elbow Method does not provide a clear K.

📌 DBSCAN and HDBSCAN, Density-Based Clustering

Density-based clustering is a powerful method used to **detect clusters of arbitrary shapes** and **identify noise (outliers)** in datasets. Unlike **K-Means**, which assumes spherical clusters and requires a predefined **K**, **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** and **HDBSCAN (Hierarchical DBSCAN)** dynamically adapt to **density variations** in data, making them useful for complex real-world clustering problems.

✔ Why use density-based clustering?

- Handles arbitrarily shaped clusters (e.g., spirals, curves).
- Works well when the number of clusters is unknown.
- Automatically identifies outliers (noise points).
- More robust than K-Means for real-world, noisy datasets.

🔹 How DBSCAN works?

DBSCAN groups points into clusters based on **local density** instead of distance from a centroid. It relies on **two parameters** to define cluster formation:

✔ **ε (Epsilon - Neighborhood Radius)** → Defines the **maximum distance** within which points are considered **neighbors**.

✔ **MinPts (Minimum Points)** → The **minimum number of points** required to form a **dense region (core point)**.

DBSCAN labels data points into **three categories**:

- **Core Points ->** Have at least MinPts neighbors within ε radius.
- **Border Points ->** Fall within the neighborhood of a core point but have fewer than MinPts neighbors.
- **Noise (outlier) Points ->** Do not belong to any cluster because they lack enough nearby points.

✔ DBSCAN Algorithm Steps

1. Select an **unvisited point** and check if it is a **core point** (has at least MinPts neighbors).
2. If it's a **core point**, create a **new cluster** and expand it by adding all density-connected points.
3. If it's a **border point**, assign it to an existing cluster.
4. If it's a **noise point**, mark it as an **outlier**.
5. Repeat until all points are **processed**.

⚠️ **DBSCAN is a single-pass algorithm** → Once a point is labeled, it does **not change** (unlike K-Means, which iterates).

Example: Clustering with DBSCAN:

> Imagine a **dataset with two crescent-shaped clusters**. K-Means would fail because **it assumes spherical clusters**, but DBSCAN successfully separates them because it recognizes **dense regions, regardless of shape**

🚀 **Strengths of DBSCAN**

- ✔ Finds **clusters of arbitrary shapes**.
- ✔ **No need to predefine K** (like in K-Means).
- ✔ **Handles outliers** by labeling them as noise.

⚠️ **Limitations of DBSCAN**

❌ **Struggles with varying densities** → The same ε may be **too small for sparse clusters and too large for dense ones**.

❌ **Sensitive to parameter tuning** → Requires careful selection of ε **and MinPts.**

🔹 How HDBSCAN works?

HDBSCAN enhances DBSCAN by **dynamically adjusting the density threshold**, making it more flexible for **datasets with clusters of varying density**.

✔ **No need to predefine ε (radius)** → HDBSCAN **automatically finds the optimal density threshold**.

✔ **Uses cluster stability** → A stable cluster **remains unchanged** when density parameters are adjusted.

✔ **Creates a hierarchy of clusters** → It builds a **tree structure (dendrogram)** and extracts the **most stable clusters**.

✔ HDBSCAN Algorithm Steps

1. Treat each point as a separate cluster (initial state).
2. **Merge clusters based on density connectivity** (reducing the density threshold).
3. Build a hierarchy (dendrogram) of clusters.
4. **Extract the most stable clusters** (those that persist across different density thresholds).

🚀 **Strengths of HDBSCAN**

✔ **No need to specify ε** (better for real-world applications).

✔ **Handles varying densities** in different regions of data.

✔ **Finds more meaningful clusters than DBSCAN**.

⚠️ **Limitations of HDBSCAN**

❌ More **computationally expensive** than DBSCAN.

❌ **Cluster stability tuning requires expert interpretation**.

🔹 Comparing DBSCAN vs. HDBSCAN

Feature	DBSCAN	HDBSCAN
Predefined Parameters?	Requires **ε & MinPts**	**No predefined ε** (adaptive density)
Handles Noise?	☑ Yes	☑ Yes
Handles Varying Densities?	❌ No (fixed density threshold)	☑ Yes (dynamic density threshold)
Cluster Shape Flexibility	☑ Arbitrary	☑ Arbitrary
Performance	Fast	Slower but more accurate

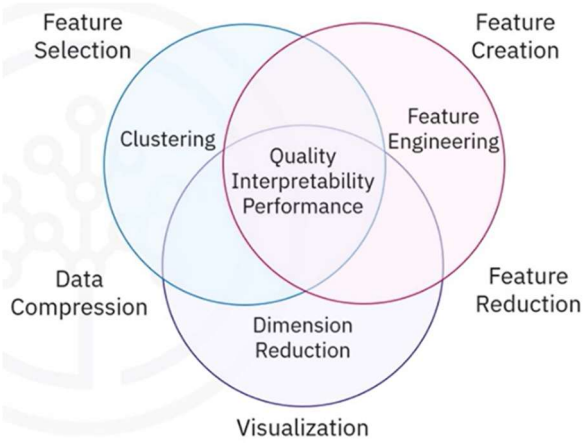
🚀 **When to Use Which?**

✔ **Use DBSCAN** when clusters have **similar density** and you can define **ε & MinPts** manually.

✔ **Use HDBSCAN** when clusters have **varying densities** and need **automatic tuning**.

📌 Dimension Reduction & Feature Engineering

Unsupervised learning techniques, including **clustering**, **dimensionality reduction**, and **feature engineering**, are essential in **data preprocessing, pattern discovery, and improving machine learning models**.



These methods allow the extraction of meaningful insights from high-dimensional and unlabeled datasets by reducing redundancy and improving computational efficiency.

This document explores:

- How clustering, dimension reduction, and feature engineering complement each other.
- The role of PCA, t-SNE, and UMAP in simplifying high-dimensional data.
- How clustering aids in feature selection and engineering.
- Comparing different dimension reduction techniques and their real-world applications.

🔹 Clustering, Dimension Reduction & Feature Engineering: How They Work Together

These three techniques serve distinct but interrelated functions in machine learning.

✔ **Clustering** → Identifies patterns and groups similar data points together.

✔ **Dimension Reduction** → Reduces the number of features while preserving important information.

✔ **Feature Engineering** → Transforms raw data into meaningful features that improve model accuracy.

🔹 How They Complement Each Other

- **Clustering helps feature selection** → Identifies **redundant features**, allowing for efficient selection of relevant ones.
- **Dimension reduction improves clustering performance** → Reduces the **noise and computational burden** of clustering algorithms.
- **Feature engineering enhances interpretability** → Creates new features that make machine learning models more accurate.

Key Insights:

When working with high-dimensional datasets, **applying dimension reduction before clustering** improves efficiency and leads to more meaningful group formations.

🔹 The Importance of Dimension Reduction

High-dimensional data poses **visualization, interpretability, and computational challenges** in machine learning.

✔ **Reduces computational cost** → Handling fewer features speeds up training and improves scalability.

✔ **Prevents overfitting** → Eliminates redundant and less informative features, making models more generalizable.

✔ **Enhances data visualization** → Helps represent high-dimensional data in 2D or 3D for better insights.

✔ **Improves clustering efficiency** → Many clustering algorithms perform poorly in high-dimensional spaces due to data sparsity.

Why do we need Dimension Reduction:

As the number of features increases, data points become **sparser**, making it harder to identify meaningful clusters. By reducing dimensions while retaining key information, we ensure that clustering and other machine learning techniques remain **effective and computationally feasible**.

📌 Linear Dimension Reduction: PCA

🔹 Understanding PCA

PCA is a **linear transformation** technique that projects high-dimensional data into a lower-dimensional space while **preserving variance**. Instead of removing features, PCA reorganizes the data into new uncorrelated features called principal components.

✔ Assumes dataset features are linearly correlated.

✔ Minimizes information loss while simplifying data structure.

✔ Transforms features into uncorrelated principal components.

✔ The first few principal components capture the most variance in the data.

📌 **Key Benefits of PCA**

- Retains **important patterns in data** while reducing complexity.
- Helps **remove noise** by discarding low-variance components.
- Improves **clustering performance** by making distances between points more meaningful.

📌 **Limitations of PCA**

❌ **Only captures linear relationships** → If data is **nonlinear**, PCA may not perform well.

❌ **Not useful for datasets with low feature correlation** → If features are **not correlated**, PCA won't be effective in reducing dimensionality.

❌ **Loses interpretability** → The transformed components **do not have a direct meaning**, unlike original features.

🔹 Understanding PCA's Dependence on Correlation

PCA works by finding new **orthogonal axes (principal components)** that capture the **maximum variance** in the data. It assumes that **original features are correlated**, so that a few principal components can **explain most of the variance**.

✔ **When features are highly correlated** → PCA finds principal components that effectively **reduce redundancy** and **compress data** while retaining variance.

❌ **When features are uncorrelated** → Each feature already represents **independent information**, so PCA **cannot combine them meaningfully** into fewer components.

If the original features have **low correlation**, the principal components will **not capture much variance**, and PCA will behave similarly to **a simple rotation of the feature space** without meaningful dimensionality reduction.

Effects of Applying PCA on Uncorrelated Features:

1. **Each principal component captures roughly equal variance** → No component dominates in explaining the structure of the data.
2. **PCA fails to provide significant dimensionality reduction** → You may still need as many components as original features.
3. **Original feature importance is lost** → PCA mixes features in ways that might make interpretation harder without improving efficiency.

✔ **If features are weakly correlated**, applying PCA **will not significantly improve model performance** and may result in unnecessary complexity.

🔹 When and When Not to use PCA

Scenario	Effectiveness of PCA	Why?
Highly correlated features	☑ Effective	PCA removes redundancy and reduces dimensions.
Moderately correlated features	⚠️ Partially effective	PCA may still help, but the reduction may not be drastic.
Uncorrelated features	❌ Not effective	PCA cannot combine independent features meaningfully.

🚀 Key Takeaway:

PCA is **most useful when applied to datasets where features show significant correlation**.

If features are **independent and uncorrelated**, other dimensionality reduction techniques like **feature selection or autoencoders** may be more appropriate.

If PCA is ineffective due to low correlation, consider the following alternatives:

✔ Feature Selection Techniques

- Remove irrelevant or redundant features using methods like:
  - **Variance Thresholding** → Drops low-variance features.
  - **Mutual Information** → Selects the most informative features for the target.
  - **Recursive Feature Elimination (RFE)** → Eliminates less important features iteratively.

✔ Non-Linear Dimensionality Reduction

- **t-SNE or UMAP** → These algorithms can capture non-linear patterns that PCA misses.

✔ Clustering-Based Feature Grouping

- Group features into clusters and retain one representative feature per cluster.

📌 Non-Linear Dimension Reduction: t-SNE vs. UMAP

Unlike PCA, which assumes **linear relationships**, **t-SNE and UMAP** capture **nonlinear structures** in data.

🔹 T-SNE (T-Distributed Stochastic Neighbor Embedding)

t-SNE is a **nonlinear embedding technique** that maps high-dimensional data into a **low-dimensional space**, focusing on preserving local relationships.

✔ Works well for **clustering complex datasets** (e.g., image recognition, NLP).

✔ Keeps **similar points close together**, ensuring good cluster visualization.

✔ Provides **better separation of groups** compared to PCA.

Limitations of t-SNE:

❌ **Computationally expensive** → t-SNE requires extensive computation and doesn't scale well to large datasets.

❌ **Sensitive to hyperparameters** → The choice of perplexity and learning rate can **drastically alter results**.

❌ **Focuses only on local structure** → Does not maintain **global structure**, meaning distant clusters may not appear where expected.

❌ **Not suitable for predictive modeling** → Since it distorts distances, t-SNE is not ideal for downstream machine learning tasks.

🔹 UMAP (Uniform Manifold Approximation and Projection)

UMAP is another **nonlinear dimensionality reduction technique** that balances **local and global structure retention** while being computationally more efficient.

✔ Captures **both local and global relationships** better than t-SNE.

✔ Works well for **large datasets** and is faster than t-SNE.

✔ Preserves **cluster structure** for improved machine learning performance.

Limitations of UMAP:

❌ **Difficult to interpret** → The transformed axes **do not have a clear meaning**, unlike PCA.

❌ **Parameter sensitivity** → While more stable than t-SNE, choosing the right **n_neighbors** and **min_dist** values affects clustering quality.

❌ **Can over-cluster data** → UMAP may create clusters that **do not exist in the original data**.

❌ **Reproducibility issues** → Results can slightly vary across different runs unless a fixed random seed is set.

🔹 **Comparison of PCA, t-SNE, and UMAP**

Algorithm	Strengths	Limitations
PCA	Reduces dimensions efficiently, fast	Struggles with nonlinear data
t-SNE	Preserves local structure, good for clustering	Slow, sensitive to tuning, distorts global structure
UMPA	Scales well, retains both local & global structure	Somewhat difficult to interpret, can over-cluster

📌 Clustering for Feature Selection & Engineering

Clustering techniques can help not only with **grouping data points** but also with **improving feature selection and engineering**.

Feature Selection with Clustering

✔ Identifies **redundant features** by grouping similar ones together.

✔ Helps remove **highly correlated features**, reducing dimensionality.

✔ Enhances model interpretability by keeping only the most useful features.

Clustering-Based Feature Selection Approach

- Clustering algorithms can be used to group features into **similar categories**.
- Features belonging to the **same cluster are often redundant**, and only one from each group needs to be retained.
- This **simplifies datasets** while preserving their predictive power.

Module 4
Best Practices for Ensuring Model Generalizability

📌Cross-Validation and Advanced Model Validation Techniques

Model validation is the process of assessing how well a machine learning model performs on **unseen data**. It is essential to ensure that the model is not overfitting the training data and is capable of **generalizing** to new inputs. In practice, model validation allows us to select and fine-tune models while preserving their predictive integrity on future, real-world data.

During model development, machine learning practitioners optimize the model by adjusting **hyperparameters**. However, if this tuning is done using the **test set**, it leads to a critical issue known as **data snooping**.

Instead of relying solely on the model's performance during training, validation strategies assess how well the model performs on **data it hasn't seen before**. This makes model validation essential for:

- Choosing between different model configurations.
- Tuning hyperparameters responsibly.
- Ensuring model robustness in real-world deployment.

🔹 Understanding Data Snooping and Data Leakage

Checking performance on the test data before you are done optimizing your model is called **data snooping**, a form of what's known as **data leakage**.

Data snooping refers to any situation where information from the **test data** leaks into the training or model selection process. This contamination often leads to **over-optimistic results** that don't hold up in production.

Example: If you tune model hyperparameters based on the test set, you are essentially allowing the model to "peek" at future data. This gives a **false sense of performance**.

Why is it dangerous?

- Your model becomes tailored to a specific test set rather than a general solution.
- Performance estimates become unreliable.
- Deployment results are disappointing.

🔹 Proper Validation Workflow: Training, Validation, Testing

What can you do to validate your model to ensure it doesn't overfit itself to your test data? - we need to decouple model tuning from the final evaluation.

Validation means tuning your model on the training data, but only testing it on unseen test data once you are satisfied that it is well trained. There is no snooping involved.

To avoid data snooping and ensure fair evaluation, datasets **ideally should** be divided into **three distinct sets:**

1. Training Set

- Used to train the model and tune hyperparameters.
- The model learns patterns from this portion.

2. Validation Set

- Used to evaluate model performance during development.
- Helps in tuning hyperparameters or selecting between models.
- Never used for final evaluation.

3. Test Set

- Set aside until final model selection is complete.
- Used to assess generalization performance after training and tuning.
- Provides an unbiased estimate of model accuracy on unseen data.

💡 **This structure ensures each data subset has a specific purpose, preventing data leakage and improving model reliability.**

⚠️ If you split your data into **just a training and a test set**, and use the test set to select the best hyperparameters, you're essentially allowing your model to **"see" the test data during training**. This leads to **overfitting the test set**, **data snooping**

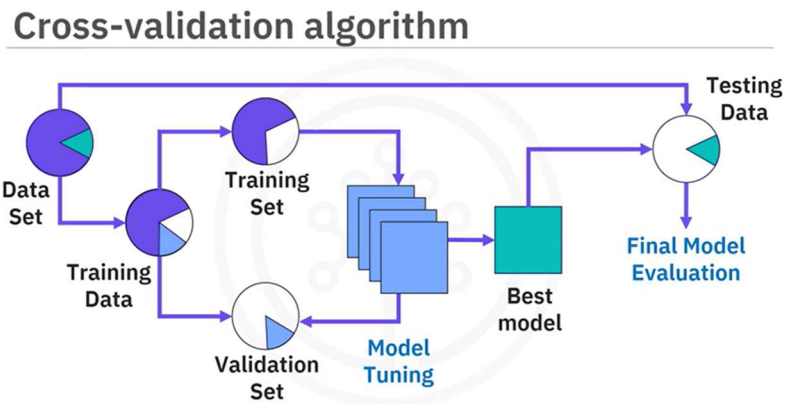
🔹 Cross-Validation (CV)

Cross-validation is a **critical model validation strategy** that helps ensure your model generalizes well to unseen data, especially during the **hyperparameter tuning** stage. It addresses several challenges that come with simpler validation techniques like train-test split — particularly the risks of **overfitting** and **data snooping**.

Cross-validation is an advanced model validation approach where the dataset is **split multiple times in a structured way** to evaluate model performance more thoroughly and prevent overfitting.

It involves:

- Dividing the training data into **training and validation subsets multiple times**.
- Training and evaluating the model on each combination.
- Aggregating the performance scores to get a more **reliable estimate** of how the model will behave on unseen data.



This way, the model **never "sees" the final test set** during training or tuning.

🔸 ***K-Fold Cross-Validation (CV)***

K-Fold CV is the most commonly used form of cross-validation.

🔍 How it works:

1. The dataset is **split into K equal parts** (called "folds").
2. For each of the K folds:
    - One-fold is held out as the **validation set**.
    - The remaining **K-1 folds** are used to **train the model**.
    - The model is evaluated on the validation fold.
3. This process is **repeated K times**, each time with a different fold as the validation set.
4. After all K iterations, the **average of the scores** (e.g., accuracy, F1, RMSE) is calculated to assess model performance.

☑ Benefits of K-Fold CV

- **Better data usage**: Every data point is used **for both training and validation**, which is especially useful when datasets are small.
- **Robust evaluation**: Since multiple validation sets are used, the performance metrics are **less sensitive to data partitioning**.
- **Prevents overfitting**: Helps in **smoothing out noise** or patterns that may exist only in a particular data split.
- **More reliable hyperparameter tuning**: Ensures that chosen model configurations generalize well.

📌 **K=5 or K=10** are common and effective choices.

Stratified Cross-Validation (for Classification)

In classification problems with **imbalanced class distributions**, standard K-Fold may lead to folds where **some classes are underrepresented**. This skews evaluation results.

🧩 **Solution: Stratified K-Fold CV**

- Ensures that each fold **preserves the class distribution** of the full dataset.
- Prevents **evaluation bias**, especially for metrics like precision, recall, or F1 score.

This is **critical for imbalanced datasets**, where naive splitting may give unreliable performance metrics.

Regression and Skewed Target Variable

In **regression tasks**, instead of class imbalance, a common issue is **skewed distributions** in the target variable. Many models assume the target variable is **normally distributed**.

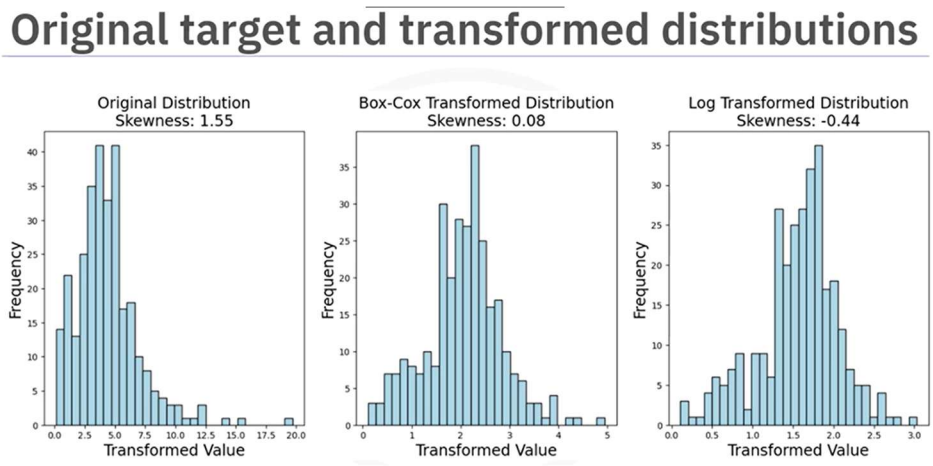
🧩 **Solutions:**

- **Logarithmic Transformation**: Applies a log function to compress large values.
- **Box-Cox Transformation**: More flexible; can normalize data using a power transformation.

These transformations:

- Reduce **skewness**.
- Make **residuals more symmetric**.
- Improve **linear model fit**.
- Stabilize **variance**.

💡 After transformation, models like **linear regression** become more effective at capturing relationships in the data.



- ***Why Cross-Validation Improves Generalizability***
  - It exposes the model to **different subsets of the training data**, helping it **learn more generalizable patterns**.
  - Models that consistently perform well across folds are more likely to **generalize to real-world, unseen data**.
  - It reduces the risk of **selecting a model that only performs well on one particular split**.

📌Regularization in Regression and Classification

Regularization is a **fundamental technique in regression modeling** aimed at **enhancing model generalizability** and **reducing overfitting**. In regression, overfitting occurs when a model learns not only the underlying patterns but also the noise in the training dataset, resulting in poor performance on unseen data.

Regularization addresses this by **introducing a penalty** that discourages the model from fitting too closely to the training data, thereby **promoting simpler and more robust models**.

What is Regularization?

Regularization modifies the **loss function** of a regression model to include a **penalty term** that penalizes large coefficient values (also called weights). The idea is to **constrain** the model so that it cannot rely excessively on any one feature, especially if that feature's predictive power is questionable or if it captures noise.

🔧 **Regularized Loss Function:**

$$Regularized\ Loss = MSE + \lambda * \text{Penalty}$$

- **Mean Squared Error (MSE):** The typical loss function in regression.
- **λ (Lambda):** Regularization parameter that controls how much penalty is added.
- **Penalty:** Depends on the type of regularization (L1 or L2 norm).

Regularization helps control **model complexity** and **stabilizes learning**, especially in datasets with a large number of features or when features are highly correlated.

🔹 Linear Regression (No Regularization)

In ordinary least squares (OLS) linear regression:

- The goal is to find the best-fitting straight line (or hyperplane) that minimizes the MSE.
- The model assumes that predictions are a **linear combination** of input features:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- This method **does not apply any penalty** on the coefficients.

⚠️ **Limitations of Linear Regression:**

- **Prone to overfitting**, especially with many features.
- Sensitive to **outliers** and **multicollinearity**.
- Coefficients can become large, unstable, or misleading in noisy datasets.

🔹 Ridge Regression (L2 Regularization)

Ridge regression introduces an **L2 penalty**, which is the **sum of squared coefficients**:

$$Penalty = \sum_{j=1}^{n} \theta_j^2$$

⚙️ **Key Characteristics:**

- Shrinks all coefficients **towards zero** but never completely eliminates any.
- Useful when **all features are informative** but possibly collinear.
- Helps reduce the model's sensitivity to **minor fluctuations in data**.

☑ **Advantages:**

- Addresses **multicollinearity**.
- Distributes influence across features.
- Performs well when **many features contribute small effects**.

🔹 Lasso Regression (L1 Regularization)

Lasso regression applies an **L1 penalty**, which is the **sum of absolute values of coefficients**:

$$Penalty = \sum_{j=1}^{n} |\theta_j|$$

⚙ **Key Characteristics:**

- Encourages **sparsity**: some coefficients are driven to **exactly zero**.
- Effective for **feature selection**—retains only the most important features.
- Ideal for **high-dimensional** datasets with irrelevant or redundant features.

☑ **Advantages:**

- **Reduces model complexity** automatically.
- Improves **interpretability** by eliminating unimportant predictors.
- Robust in situations where **only a subset of features is truly predictive**.

🔹 When to Use Ridge or Lasso?

Scenario	Recommended Approach
Many features, all potentially useful	**Ridge Regression**
Sparse data, irrelevant features present	**Lasso Regression**
High collinearity among features	**Ridge Regression**
Feature selection is desired	**Lasso Regression**
Limited data, risk of overfitting	**Lasso or Ridge**

Both techniques are not mutually exclusive. A hybrid approach, Elastic Net, combines both L1 and L2 penalties and is used when you want both feature selection and shrinkage.

🔹 Performance Under Different Data Conditions

Regularization techniques like Ridge and Lasso are designed to improve model generalization, especially in challenging data environments. Their effectiveness varies depending on two major conditions:

1. **Sparsity of the Data:**

   Sparsity refers to how many of the input features (coefficients) are truly relevant to predicting the outcome.

   - Sparse Coefficients:
     - Most features have little or no influence on the target variable.
     - Only a few features carry meaningful signals.

   🧠 How Models Perform:

Model	Behavior
Lasso	Excels in sparse settings. It can zero out irrelevant coefficients, effectively performing feature selection.
Ridge	Tries to shrink all coefficients but does not eliminate any. It's less effective when many features are irrelevant.
Linear Regression	Often overfits. It gives weight to all features, including noise, leading to poor generalization.

2. **Signal-to-Noise Ratio (SNR):**

   SNR compares the **strength of the true signal** in the data to the **random noise**.

   - **High SNR:**
     - The useful signal is strong and easy to distinguish from noise.
     - Model has a clearer path to learning meaningful patterns.

   🧠 How Models Perform:

Model	Behavior
Lasso	Accurately identifies relevant features and sets unimportant ones to 0.
Ridge	Slightly over-smooths but still fits well.
Linear Regression	Performs decently, as noise isn't strong enough to cause significant overfitting.

- **Low SNR:**
  - The data is **very noisy**, making it hard to detect real patterns.
  - There's a higher risk of overfitting noise instead of learning the actual signal.

🧠 How Models Perform:

Model	Behavior
Lasso	Performs the best overall. It ignores irrelevant/noisy features, and even in low-SNR environments, it focuses only on strong signals.
Ridge	Better at controlling variance; shrinks coefficients to **avoid overfitting noise**.
Linear Regression	Performs **very poorly**. It reacts strongly to noise, inflating and distorting coefficients.

☑ Takeaways

☑ Regularization is a core component of modern machine learning pipelines.

☑ It directly influences generalization, interpretability, and stability.

☑ Understanding the structure of your data (sparse vs. dense, noisy vs. clean) is key to selecting the right method.

☑ In practice, hyperparameter tuning (especially λ) using techniques like cross-validation ensures the optimal level of regularization.

☑ **Lasso** is ideal when you believe that **only a few features are important**. It thrives in **sparse, noisy** data.

☑ **Ridge** is effective when **all features may contribute** and the goal is to **stabilize coefficients**, especially under **moderate noise**.

☑ **Linear Regression** can perform well **only when the data is clean and signal is strong**. Otherwise, it's prone to **overfitting** and poor generalization.

📌 Data Leakage and Modeling Pitfalls

In the machine learning workflow, ensuring that models generalize well to unseen data is as critical as building accurate models. However, several challenges—such as **data leakage**, **incorrect validation**, and **misleading feature importance**—can compromise a model's reliability.

🔹 Data Leakage

Data leakage occurs when information that would not be available in real-world production scenarios is **inadvertently included in the training data**. This leads to **unrealistically high-performance during training or validation**, and severe **underperformance when deployed**.

🧪 Example:

You build a housing price prediction model and include a feature derived from the **average price of the entire dataset**, which includes future data. This information won't be available at inference time, and the model won't replicate its performance in production.

📉 Consequences:

- Artificially high accuracy during model training/validation.
- Misleading test scores.
- Poor real-world generalization

💡 Best Practices for preventing Data Leakage:

- **Avoid using features** that depend on the **entire dataset** (e.g., global averages).
- Ensure **no future or target-derived data** leaks into the training process.
- Maintain strict **separation between training, validation, and test sets**.
- Use proper **data pipelines**, fitted independently on training folds during cross-validation.
- Be cautious with **feature engineering**, especially on temporal or sequential datasets.

🔹 Data Snooping

Data snooping (a form of data leakage) occurs when a model **has access to information from the test set** or data that it **should not have seen**. This commonly happens during hyperparameter tuning or improper preprocessing.

📌 **Example:**

Selecting hyperparameters based on test set results leads to the model being tuned **for that specific test set**, thereby **invalidating** the evaluation.

🧰 **Mitigation:**

- Use **K-Fold Cross-Validation** for hyperparameter tuning.
- Always evaluate final performance on a **held-out test set**.
- Perform **data preprocessing within cross-validation folds**, not globally.

🔹 Time-Series Data & Sequential Validation

When working with Time-Series Data, special care is needed, because in time-dependent datasets (e.g., stock prices, weather, IoT sensors), the order of observations is critical. Random splitting violates the temporal structure, causing the model to "see the future" during training—an indirect form of data leakage.

💡 **Best Practice: Time-Series Cross-Validation**

Use **TimeSeriesSplit**, a cross-validation strategy that:

- Preserves the chronological order of observations.
- Ensures each training set **precedes** the validation set.
- Expands the training window and slides the test window forward.

Each fold uses only past data for training and future data for testing, mimicking real-world deployment.

☑ **Summary Steps for Time-Aware CV:**

1. Avoid train_test_split for time-series.
2. Use TimeSeriesSplit from scikit-learn.
3. Ensure **no shuffling**, and all preprocessing steps (e.g., scaling, PCA) are applied **within each fold**.
4. Wrap your transformations and model in a Pipeline, and pass it to GridSearchCV with cv=TimeSeriesSplit(...).

🔹 Feature Importance Interpretation Pitfalls

Feature importance can be a powerful tool for understanding model behavior, diagnosing errors, and improving performance. However, interpreting importance **incorrectly or out of context** can lead to **false conclusions** or misguided model adjustments.

Here are the most **common pitfalls**:

1. **Redundant of Highly Correlated Features**

   When features are highly correlated (multicollinearity), their importance gets **"split"** across them, leading to the **illusion that each is less important than it actually is**.

   - In **linear models**, this causes **instability** in coefficient estimation.2
   - In **tree-based models**, different splits across trees may favor one over the other arbitrarily.

   🛠️ **Mitigation:**

   - Use **correlation matrices** or **variance inflation factor (VIF)** to detect redundancy.
   - Consider **dropping or combining** highly correlated features.
   - Use models robust to multicollinearity (e.g., **Lasso**).

2. **Feature Scaling Sensitivity**

   Some algorithms, especially linear models and distance-based models (like KNN or SVM), are **sensitive to feature scale**.

   - Features with **larger numerical ranges** can dominate importance scores, **even if they're not more informative**.
   - In **linear regression**, unscaled features lead to **misleading coefficient magnitudes**, misrepresenting feature impact.

   🛠️ **Mitigation:**

   - Always apply **standardization or normalization** before training scale-sensitive models.
   - Use **Pipelines** to ensure preprocessing is applied consistently during cross-validation.

3. **Assuming Correlation Implies Causation**

   A feature being important in a model **does not mean it causes the outcome**.

   - Example: An air conditioner usage feature might appear important in predicting electricity usage—but it's a **proxy for temperature**, not the causal driver.
   - Mistaking correlation for causation can lead to **invalid interventions** (e.g., assuming that changing a correlated feature will improve outcomes).

   🛠️ **Mitigation:**

   - Use **domain knowledge** to interpret importance.
   - Consider **causal inference techniques** when decisions rely on understanding cause-effect relationships.

4. **Overlooking Feature Interactions**

   Some models rank features **individually**, failing to account for **how features work together**.

   - Example: Neither height nor weight may be useful alone, but their **ratio (BMI)** may strongly predict health outcomes.
   - **Linear models** especially fail to detect interactions unless explicitly engineered (e.g., via polynomial features or interactions).

   🛠️ **Mitigation:**

   - Use **models that capture interactions**, such as **tree ensembles (Random Forest, XGBoost)** or **neural networks**.
   - **Engineer interaction features** if using linear models.
   - Apply **SHAP (SHapley Additive exPlanations)** to understand feature interactions in complex models.

5. **Model-Specific Interpretations**

   Feature importance is **model-specific**: what's important in one model may not be in another.

   - A feature may be unimportant in a **linear model**, but crucial in a **nonlinear one**.
   - Comparing importance **across models** without accounting for modeling assumptions leads to **false conclusions**.

🛠️ **Mitigation:**

- Compare models **within their own context**.
- Use **agnostic model explanation tools** like **permutation importance** or **SHAP** to compare importance across different models fairly.

6. **Miss-using Importance for Feature Selection**

   Just because a feature has low importance in one model doesn't mean it's safe to remove.

   - It may be:
     - Important in **combination** with other features.
     - Useful for **future data distributions** not yet seen.
     - A **key driver** of regularization in Lasso or Ridge.

   🛠️ **Mitigation:**

   - Avoid dropping features **only** based on one model's importance scores.
   - Evaluate feature subsets using **cross-validation** or **recursive feature elimination (RFE)**.

📌 Evaluating Machine Learning Models

Supervised learning evaluation establishes how well a machine learning model can predict the outcome for unseen data.

It is essential for understanding model effectiveness and involves comparing model predictions to ground truth labels.

During training, the model tries to optimize predictions based on one or more evaluation metrics.

After training, the model is again evaluated to estimate how well it can generalize to unseen data.

Supervised learning evaluation is essential in both the training and testing phases.

Evaluating the performance of a machine learning model is **crucial** to understanding its ability to **generalize** to new data.

📌 Train-Test Split

The Train-Test Split technique is used to estimate how well a machine learning model will perform on unseen data.

📌 Why Split the Data?

✔ **Prevents Overfitting →** Ensures the model doesn't memorize the training data.

✔ **Simulates Real-World Predictions →** Tests the model's ability to generalize.

✔ **Avoids Data Snooping →** Prevents the model from learning patterns it shouldn't have access to.

📌 How It Works?

1. **Training Set →** Typically 70-80% of the dataset is used to train the model.
2. **Testing Set →** The remaining 20-30% is reserved for evaluation.

💡 Best Practice:

- Ensure the dataset is randomly shuffled before splitting.
- Use stratified sampling when dealing with imbalanced classes.

📌 Classification Metrics and Evaluation Techniques

A classification model predicts **categorical labels**. To assess how well the model performs, we use key **evaluation metrics**:

🔹 Accuracy:

Definition: The ratio of correctly predicted observations to the total number of observations

Formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total Observations}}$$

☑ **Useful when** → Classes are **balanced** (i.e., similar numbers of positive and negative cases).

❌ **Not ideal when** → There is **class imbalance** (e.g., rare disease detection, fraud detection).

🔹 Confusion Matrix:

Definition: confusion matrix is a table that **compares the predicted class labels to the actual labels**. It helps visualize where the model is making **correct and incorrect predictions**.

Actual \ Predicted	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

- **True Positives (TP)** → Correctly predicted **positive** instances.
- **True Negatives (TN)** → Correctly predicted **negative** instances.
- **False Positives (FP)** → Incorrectly predicted **positive** instances (Type I Error).
- **False Negatives (FN)** → Incorrectly predicted **negative** instances (Type II Error).

🔹 Precision: The Accuracy of Positive Predictions:

Definition: Precision measures how many of the predicted **positive** instances are **actually positive**.

Formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

☑ **Use Precision when** → **False Positives** are **costly** (e.g., spam detection, recommending irrelevant ads).

🔹 Recall (Sensitivity): Identifying Actual Positives:

Definition: Recall measures how many of the **actual positive instances** were **correctly predicted**.

Formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

☑ **Use Recall when** → **False Negatives** are **critical** (e.g., missing a cancer diagnosis, fraud detection).

🔹 F1 Score: The balance between Precision and Recall:

Definition: The **F1 Score** is the harmonic mean of **Precision and Recall**, providing a single measure of model effectiveness.

Formula:

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

☑ **Use F1 Score when** → Both False Positives and False Negatives are equally important.

🔹 Choosing the right metric

Scenario	Best Metric	Reason
Balanced Classes	Accuracy	Works well when all classes have similar proportions.
Class Imbalance	Precision or Recall	Accuracy may be misleading if one class dominates.
Medical Diagnosis	Recall	False Negatives are more dangerous than False Positives.
Spam Detection	Precision	False Positives should be minimized.
General Classification	F1 Score	Balances Precision & Recall when both are important.

📌 Regression Metrics and Evaluation Techniques

Evaluation is critical because:

- It quantifies how well the model generalizes to **unseen data**.
- It helps identify **underfitting, overfitting**, or potential **bias** in predictions.
- It guides **model selection, hyperparameter tuning**, and **interpretability**.

When a model makes predictions, the difference between the predicted and actual values is known as **error**. A well-performing regression model should minimize these errors.

Regression metrics quantify **how far off** a model's predictions are from the actual values. Each metric gives **different insights** into error magnitude, consistency, and model performance.

🔹 MAE (Mean Absolute Error):

- Measures the average absolute difference between predicted values and actual values.
- Every error is treated equally, regardless of direction or size.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | \hat{y}_i - y_i |$$

🔍 **When to Use:**

- When **outliers are expected** but should not dominate the error measure.
- When **interpretability in actual units** (e.g., dollars, degrees) is important.

🔹 MSE (Mean Squared Error):

- Squares the errors before averaging, thus **penalizing large deviations more heavily**.

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

🔍 **When to Use:**

- When **larger errors are unacceptable** and need to be discouraged during training.
- Ideal for algorithms that **optimize squared loss**, such as linear regression.

🔹 RMSE (Root Mean Squared Error):

- The square root of MSE.
- Expressed in the same units as the target variable, making it more intuitive than MSE.

$$RMSE = \sqrt{RMSE}$$

🔍 **When to Use:**

- When you want an **error measure in the same unit** as your prediction target.
- Useful when large errors are very costly and should be emphasized.

🔹 R^2 Score (Coefficient of Determination):

- Measures the proportion of variance in the target variable that the model explains.
- Ranges from 0 (no explanatory power) to 1 (perfect prediction). Negative values indicates that model is **worse than just predicting the mean**.

$$R^2 = 1 - \frac{Unexplained\ Variance}{Total\ Variance}$$

🔍 **When to Use:**

- To assess the goodness of fit.
- To explain to stakeholders how much of the outcome your model captures.

ℹ️ Note: R^2 assumes **linear relationships** between features and target. In **nonlinear models**, it can be misleading if used in isolation.

📌 Evaluation insights

🔹 Combine Multiple Metrics for a Full Picture

- **MAE** gives an average error magnitude but is forgiving of outliers.
- **MSE/RMSE** emphasize larger errors, useful when **large mistakes are expensive**.
- **R^2** tells how well the model captures the data structure but doesn't reveal **error direction** or **bias**.

✔ Always **use multiple metrics together**, as no single number captures the whole performance landscape.

🔹 Use Residual Analysis to Detect Bias

A model might have strong metrics but still fail in specific ranges of the target variable. This is where residual plots become essential:

- Random scatter around zero → well-behaved model.
- Curved or sloped patterns → the model misses a nonlinear relationship.
- Systematic over/under-prediction in high or low ranges → distributional bias.

✔ Residual plots help identify where and why the model struggles, allowing targeted improvements.

🔹 High R^2 Can Be Deceptive

- A high R^2 (e.g., 0.85) doesn't mean the model is **accurate everywhere**.
- It could just mean that the model predicts the **mean trend well**, but fails on the **extremes** or **outliers**.

✔ Visualizations (actual vs predicted, residuals) are crucial for verifying that high R^2 reflects **true performance**.

🔹 Systematic Error Across Ranges

Regression models often exhibit **non-uniform accuracy**:

- **Underprediction of high values** may indicate inability to extrapolate.
- **Overprediction of low values** might occur when the model is too biased toward the dataset mean.

✔ Address this by:

- Engineering features that **capture edge-case behavior**.
- Transforming the target variable (e.g., log, Box-Cox).
- Choosing a **model with more flexibility**, such as ensembles or gradient boosting.

🔹 Interpreting Feature Importance

When using models like **Random Forests**:

- **Feature importance** indicates **how much a feature reduces prediction error**, not its **causal effect**.
- Correlated features may **share importance**, which can dilute interpretability.

✔ Use a **correlation matrix** to confirm whether features are redundant or overlapping in information.

✔ Feature selection improves **model efficiency and robustness**.

🔹 Outliers and Clipping Impact Metrics

- Models trained on datasets with **target caps or skew** (e.g., price capped at $500,000) often underperform on **high-end predictions**.
- These effects are visible in residuals and degrade R^2 and RMSE selectively.

✔ Recognize when the **target distribution itself is biased or censored**, and adjust model goals or training data accordingly.

📌Evaluating Unsupervised Learning, Heuristics and Techniques.

Evaluating unsupervised learning models is a different challenge compared to supervised ones. In supervised learning, we have labeled data to compare our predictions against — we know the correct answers. But in unsupervised learning, like clustering or dimensionality reduction, there's no ground truth. These models try to **find patterns or groupings on their own**, which means we need different strategies to assess whether the patterns found are actually meaningful.

The purpose of unsupervised models is to **detect useful structure in data** — for example, identifying clusters or reducing dimensions in a way that keeps the data informative. Evaluation helps us answer questions like:

- Do the clusters represent real patterns in the data?
- Are similar data points grouped together consistently?
- When we reduce the number of dimensions, are we still keeping the important relationships between points?

Since we don't have labels to directly measure against, we need to use tools like **heuristics**, **internal and external metrics**, **visualizations**, or **domain knowledge** to assess the quality of the results.

🧠 Stability and Generalization – Key Concepts

A key idea in unsupervised model evaluation is **stability** and **generalization**.

✔ **Stability:** A model produces similar results even when the dataset is slightly changed. For example, if you remove a few rows or shuffle the data, and the clusters still look mostly the same, that means the model is stable and reliable.

✔ **Generalization:** How well the model's discovered patterns hold up on new or different data. A good clustering model, for example, should form similar groupings even if the data varies a little. This is crucial for trusting the model's insights, especially when we're applying it to real-world, messy data.

Internal Evaluation Metrics

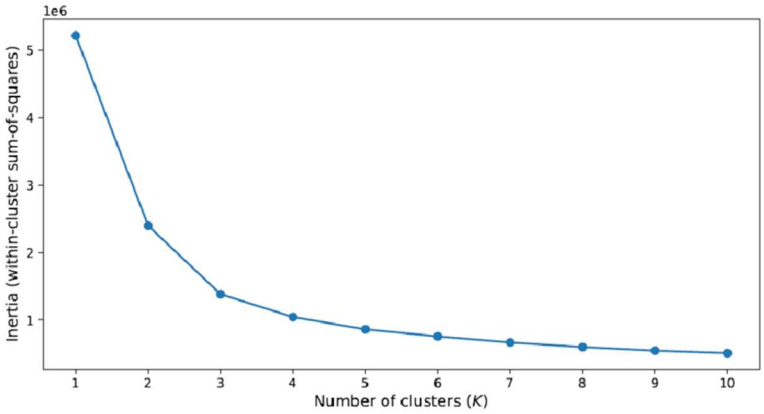
These metrics use only the data and the clustering structure (no ground truth labels).

1. **Inertia:**

   Measures the **sum of squared distances** between data points and cluster centers (used in k-means).

   ✔ Lower inertia indicates **tighter clusters**, but decreases as **k increases** — it needs to be interpreted with caution.

   ✔ Score can be plotted for different k values, and from it we can see how this score varies when k (number of clusters) change. Can be used to obtain the optimal number of clusters.



2. **Silhouette Score:**

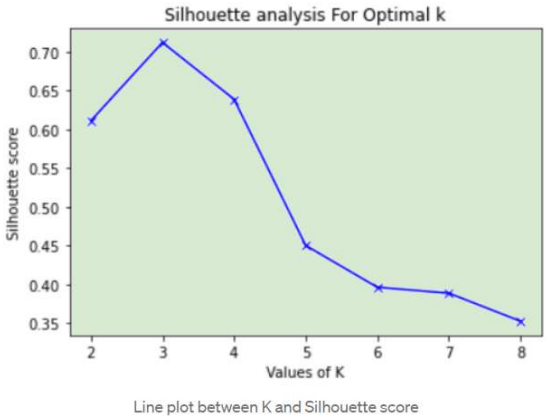
   Measures how close a point is compared to the rest of points that belongs to the same cluster, in relation to how close the same point is to the point in the nearest cluster.

   It compares the two distance measures to see how similar a point is to its own cluster, compared to others.

   ✔ Score ranges from **-1 to 1** (higher values mean better-defines clusters).

   ✔ Great for **visualizing** and **validating** cluster **compactness** and **separation.**

   ✔ Score can be plotted for different k values, and from it we can see how this score varies when k (number of clusters) change. Can be used to obtain the optimal number of clusters.



Line plot between K and Silhouette score

3. **Calinski-Harabasz Index (Variance Ratio Criterion):**

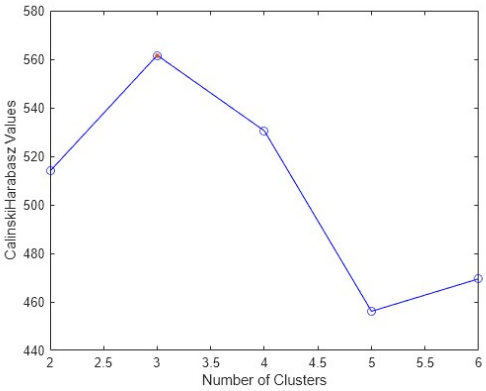
   Measures the relationship, or ratio, of between-cluster dispersion (external distance) to within-cluster dispersion (internal distance).

   ✔ A well-defined cluster structure has a **high between-cluster dispersion** and **low within-cluster dispersion**.

   ✔ Scales well and is **computationally efficient**, making it ideal for larger datasets.

   ⚠ Can sometimes **favor models with more clusters**, so best used in combination with other metrics.

   ✔ Score can be plotted for different k values, and from it we can see how this score varies when k (number of clusters) change. Can be used to obtain the optimal number of clusters.



4. **Davies-Bouldin Index:**

   Measures **intra-cluster similarity** relative to **inter-cluster separation**.

   ✔ Lower values are better (les overlap between clusters).

5. **Dunn Index:**

   Ratio of the **smallest distance between cluster centers** to the **largest intra-cluster distance.**

   ✔ **Higher values** mean better-defined, more isolated clusters.

   ✔ Sensitive to noise but provides strong insight into compactness and separation.

🔹 External Evaluation Metrics

External evaluation metrics are used when **true labels are available**, even though the learning process itself is unsupervised. These metrics help determine **how well the discovered clusters align with the actual class labels**.

They **compare the predicted cluster assignments** with the known categories (e.g., customer segments, document topics, species types), and provide **quantitative insight** into clustering quality based on **ground truth**.

These metrics are **especially valuable when validating a clustering algorithm** during benchmarking or research, or in semi-supervised learning scenarios.

1. **Adjusted Rand Index (ARI):**

   Measures the **similarity between two assignments**: one from the clustering algorithm, and one from the actual labels.

   ✔ It **corrects for chance**, so random label assignments tend toward zero, while perfect alignment scores **1**.

   ✔ Negative values can occur if the clustering is **worse than random**.

   How it works:

   It looks at **pairs of samples** and checks how consistently they were placed together or separately in both the predicted and actual groupings.

   Use it when:

   ✔ You want to test **how well your clustering matches known categories**.

   ✔ Evaluating robustness against **label shuffling** or **random noise**.

2. **Normalized Mutual Information (NMI):**

   Measures the **amount of shared information** between the cluster assignments and true labels.

   ✔ The score ranges from **0 to 1**:
   - 1 = perfect match (maximum information shared).
   - **0** = completely unrelated assignments.

   How it works:

   Based on **entropy and information theory**, it quantifies how much knowing the predicted cluster helps in guessing the true class.

   Normalization ensures fair comparison even when the number of clusters and classes differ.

   Use it when:

   ✔ You want a **balanced, symmetric metric** that isn't biased by the number of clusters or label distribution.

   ✔ Comparing different clustering configurations with uneven cluster sizes.

3. **Fowlkes-Mallows Index (FMI):**

   Evaluates clustering by computing the **geometric mean of precision and recall** (based on pairwise point assignments).

   ✔ Ranges from **0 (no match)** to **1 (perfect clustering)**.

   How it works:

   Compares every pair of samples to see:

   - Whether they belong to the **same cluster** in both true labels and predicted clusters.
   - Whether they are correctly **separated or grouped**.

   Use it when:

   ✔ It is sensitive to both false positives and false negatives in clustering.

   ✔ Provides a balanced view of clustering performance when both precision and recall are important.

🔹 Dimensionality Reduction Evaluation

Dimensionality reduction is often used to **project high-dimensional data into lower dimensions** — typically for **visualization, simplification, or preprocessing before clustering**. However, reducing dimensions always carries the risk of **information loss**, so it's crucial to **evaluate how much of the original structure is preserved** in the reduced space.

Evaluating a dimensionality reduction algorithm helps answer:

- Has the algorithm retained the **most important relationships between data points**?
- Can the reduced representation still support **meaningful clustering or classification**?
- Has it preserved **global and/or local structure**?

There are three key aspects to evaluate dimensionality reduction performance:

1. **Explained Variance Ratio (for PCA)**

   - PCA transforms the data into **principal components**, which are ranked by how much **variance** (or spread) they explain from the original data.
   - The **explained variance ratio** tells you how much information is retained in each principal component.

- If the **first few components** capture most of the variance, it means the reduced data still represents the original structure well.

☑ **Useful to decide how many dimensions you can safely reduce to without sacrificing too much information.**

🔎 *Example insight*: If the first 2 components explain 90% of the variance, you can confidently reduce your data to 2D for visualization or further analysis.

2. **Reconstruction Error**

- Applies to algorithms that **encode and decode data**, like PCA and autoencoders.
- Measures how well you can **reconstruct the original data** from the reduced representation.
- A **low reconstruction error** indicates that the reduced data still contains the essential information.

☑ **Used to assess how much information was lost** during the dimensionality reduction.

🔎 *Important when using reduced data for downstream tasks (e.g., predictions or clustering).*

3. **Neighborhood Preservation**

- Especially relevant for **nonlinear techniques** like t-SNE and UMAP.
- Measures how well the **local relationships between data points** (i.e., which points are close to which) are preserved in the low-dimensional space.
- This is crucial for visualizations or when clustering follows dimensionality reduction.

☑ **Evaluates whether nearby points in high-dimensional space remain nearby after reduction.**

🔎 *Important for tasks like manifold learning, anomaly detection, and cluster visualization.*