Machine Learning - C. Sartori

Introduction to Machine Learning **5**
Data **7**
Issues on Data **7**
Types of Data 7
Challenges with Data Quality 7
Preprocessing and Data Transformation 8
Outliers and Their Impact 8
Data Quality **9**
Common Data Quality Issues 9
Techniques to Handle Data Quality Issues 9
Data Cleaning and Preprocessing 10
Impact of Data Quality on Machine Learning 10
Association Rules **11**
Frequent Itemset Generation 13
Brute-Force Approach 13
The Apriori Principle 13
The Apriori Algorithm 14
Example of Apriori Execution 14
Factors Affecting Complexity 14
Multidimensional Association Rules 14
Comparison: Mono vs. Multidimensional Rules 15
Equivalence Between Mono and Multi-dimensional Rules 15
Handling Quantitative Attributes 15
Multilevel Association Rules 15
Concept Hierarchy in Association Rules 15
Impact on Support and Confidence 15
Mining Multilevel Association Rules 16
Conclusion on Association Rules 16
Classification **17**
Classification Model 17
Classification with Decision Trees **19**
Entropy and Information Gain **20**
Learning a Decision Tree **21**
Stopping Criteria 22

Errors and Overfitting 22
Training and Test Errors 22
Overfitting in Decision Trees 22
Indicators of Overfitting 22
Causes of Overfitting 22
Impurity Functions 23
Evaluation of a Classifier 26
Training, Validation, and Test Sets 26
Empirical Error Estimation 27
Statistical Pruning of Decision Trees 27
Model Selection for a Classifier 27
Train, Test, Evaluate, Optimize 28
Train/Test Split Strategies 28
Cross-Validation: A Robust Approach 28
Hyperparameter Optimization 28
Performance Measures of a Classifier 29
Confusion Matrix 29
Key Performance Metrics 30
Multi-Class Evaluation 30
Beyond Counting: Considering Chance and Cost 30
Evaluation of a Probabilistic Classifier 31
Probabilistic vs. Crisp Classification 31
ROC Curve (Receiver Operating Characteristic) 31
Choosing the Best Threshold 32
Statistical Modeling – Naive Bayes Classifier 32
Bayes' Theorem 32
Handling Missing Values 33
Handling Numeric Values 33
Laplace Smoothing 34
Linear Classification with the Perceptron 35
The Perceptron Model 35
Perceptron Learning Algorithm 36
Support Vector Machines (SVM) 38
Main Characteristics 38
Maximum Margin Hyperplane 38
Handling Non-Linear Data: The Kernel Trick 39
Common Kernels 39
Neural Networks 40

Multi-Layer Perceptron (MLP) 40
Feedforward Neural Networks 41
Training Neural Networks 41
Training Algorithms 42
Overfitting and Regularization 43
Neural Network Architectures 43
Instance Learning - K Nearest Neighbours Classifier 44
Loss Function 48
From a Binary Classifier to Multi-Class Classification 50
One-vs-One (OvO) 50
One-vs-Rest (OvR) 50
Alternative Multi-Class Methods 51
Ensemble Methods 52
Types of Ensemble Methods 52
Forest of Randomized Trees 54
Random Forest Algorithm 55
Boosting 58
Popular Boosting Algorithms 58
Preprocessing 65
Data Type Conversions 65
Sampling 67
Types of Sampling 67
Feature Creation 68
Data Transformations 70
Imbalanced Data in Classification 72
Strategies to Handle Imbalanced Data 73
Feature Selection 75
Dimensionality Reduction 77
The Scikit-Learn Solution for Feature Selection 79
Outlier-Detection 81
Evaluation Metrics for Outlier Detection 82
Definition and Importance 83
Key Techniques for Outlier Detection 83
Challenges in Outlier Detection 84
Evaluation Metrics for Outlier Detection 84
Problem Description 85
Anomaly Detection 85
Statistical Approaches to Outlier Detection 87

Proximity-Based Outlier Detection..... 90
Density-Based Outlier Detection..... 92
Isolation Forest..... 93
Clustering.....95
Measuring Clustering Quality..... 96
Centroid.....96
Types of Clustering Methods.....96
K-Means Clustering.....97
Evaluation of a Clustering Scheme.....100
Hierarchical Clustering.....101
Density-Based Clustering.....103
Model-Based Clustering.....105
Final Remarks.....108
Proximity measures.....110
Similarity and Dissimilarity.....110
Distance Metrics.....112
Similarity Measures..... 114
Use Cases for Distance and Similarity Metrics.....116
Regression..... 118
Overview..... 118
Key Characteristics of Regression.....118
Applications of Regression.....118
Summary.....118
Linear Regression.....119
Multiple Regression.....121
Polynomial Regression.....124
Overfitting and Regularization..... 125
Model Selection (Comparison of Regression Models).....128
Regression-Regularized Techniques.....130
Lasso Regression.....130
Ridge Regression..... 132
Elastic Net Regression.....134

Introduction to Machine Learning

A Brief History of Information Technology

The evolution of information technology has played a fundamental role in the development of machine learning and data mining. Below are key milestones:

- **1960s:** Early data collection and the creation of databases.
- **1970s:** Development of database management systems (DBMS), improving data storage and retrieval.
- **1980s:** DBMS matured, new data types emerged, and new data access paradigms were introduced.
- **1990s:** The rise of the web, data warehousing, and the concept of knowledge discovery in databases (KDD).
- **2000s–Present:** The explosion of big data and the need for sophisticated analytics to process and interpret vast amounts of information.

From Operational Data to Analytical Insights

Initially, data was primarily used for routine operations such as inventory management, billing, and census collection. Most data was used once and then archived. However, questions arose:

- Can we use stored data to improve decision-making processes?
- Can we learn valuable insights from past data?

With growing data volumes and advanced storage capabilities, the focus shifted from mere data storage to its analysis and interpretation.

The Challenge of Big Data

The rapid increase in data generation, combined with advances in DBMS technology and affordable storage, has led to an overwhelming amount of available data. However, extracting meaningful information from this data remains a challenge. A widely cited phrase encapsulates this issue:

> "We are drowning in data but starved for information." – John Naisbitt, *Megatrends* (1982).

How to Learn from Data

1. **Statistics:** Since the 18th century, statistical methods have been used to describe and infer relationships within data. These include:
    - Descriptive statistics
    - Inferential statistics
    - Statistical models
2. **Machine Learning:** Emerging in the late 1950s, ML enables computers to learn patterns and make decisions without explicit programming. Learning approaches include:
    - Learning by being told (rule-based systems)
    - Learning from examples (pattern recognition, supervised learning)
3. **Data Mining:** Introduced in the early 1990s, data mining is a computational process aimed at discovering patterns in large, digitally stored datasets. It combines concepts from:
    - Artificial intelligence
    - Machine learning
    - Statistics
    - DBMS technology

Terminology Overview

a. **Business Intelligence (BI):** The use of analytical tools to extract business insights from large datasets.
b. **Analytics:** The process of drawing conclusions from raw data.
c. **Data Mining:** A discovery process that extracts general patterns and actionable insights from data.
d. **Machine Learning:** The use of algorithms to automate pattern extraction from data.
e. **Data Science:** A broader discipline encompassing all the above areas, requiring expertise in multiple fields.

The Virtuous Loop of Data-Driven Decision Making

With advancements in IT, data-driven decision-making has become crucial across industries. Data is no longer just stored—it is actively analyzed to drive business strategies, optimize processes, and innovate in fields like healthcare, finance, and retail.

Data

Issues on Data

Types of Data

Data can be categorized into different types based on their nature and structure:

1. **Quantitative Data:** Numerical data that can be measured and counted.
2. **Qualitative Data:** Categorical data that represents characteristics and attributes.
3. **Structured Data:** Organized data stored in relational databases with predefined schema.
4. **Unstructured Data:** Information that lacks a fixed format, such as text, images, and videos.
5. **Ordered Data:** Data that follows a specific sequence, such as spatial, temporal, and genetic sequences.
6. **Graph Data:** Data that represents relationships between entities, such as web links and molecular structures.

Challenges with Data Quality

Data is rarely perfect and often contains issues that need to be addressed before analysis. Common problems include:

- **Missing Data:** Some values may be absent due to errors in data collection or entry.
- **Inconsistent Data:** Conflicts or contradictions between data entries.
- **Duplicated Data:** Redundant records present in the dataset.
- **Incorrect Data:** Mistakes in data values that may lead to erroneous conclusions.
- **Outliers:** Data points significantly different from the rest, which can be due to errors or rare events.
- **Sparsity:** Datasets where many values are zeros or nulls, affecting analysis.

➔ **Dimensionality Issues:** High-dimensional datasets introduce computational complexity and redundancy, leading to the “curse of dimensionality”.

Preprocessing and Data Transformation

To improve the quality of data and ensure better machine learning outcomes, preprocessing steps are applied:

- **Cleaning:** Removing duplicates, correcting inconsistencies, and filling missing values.
- **Transformation:** Converting data into appropriate formats (e.g., normalization, scaling, encoding).
- **Reduction:** Simplifying data by eliminating redundant or irrelevant features.
- **Discretization:** Converting continuous values into categorical bins to simplify analysis.
- **Aggregation:** Combining multiple related features into a single representation.

Outliers and Their Impact

Outliers are data points that significantly deviate from the majority. They can be caused by:

✗ Measurement errors
✗ Rare occurrences
✗ Data corruption

Some machine learning models are more robust to outliers, but their presence can still impact the accuracy and reliability of models. Effective outlier detection techniques include:

- **Interquartile Range (IQR) Method:** Identifying values that fall outside a specified range.
- **Boxplots:** Visual representations to detect extreme values.
- **Statistical Methods:** Using standard deviation and Z-scores to identify anomalies.
- **Distance-Based Methods:** Identifying anomalies by measuring the distance from neighbors.

- **Density-Based Methods:** Detecting outliers based on sparsely populated regions of data.

Importance of Data Preprocessing

A well-prepared dataset leads to better machine learning results. The principle of **“garbage in, garbage out”** highlights the necessity of high-quality data for accurate and reliable model performance.

Data Quality

Definition and Importance

Data quality refers to the accuracy, completeness, consistency, and reliability of data. Poor data quality can lead to incorrect model predictions and misleading insights.

Common Data Quality Issues

- **Noise:** Unwanted modifications in data, such as errors in transmission or mixing human and automated interactions.
- **Outliers:** Data points that significantly deviate from others, caused by rare events or errors.
- **Missing Values:** Data that has not been collected or is not applicable to certain cases.
- **Duplicated Data:** Redundant data records, often occurring when merging datasets.
- **Inconsistencies:** Conflicting or contradictory information within the dataset.

Techniques to Handle Data Quality Issues

1. **Noise Reduction:**
    - Filtering techniques to remove unwanted variations in data.
    - Identifying and correcting errors caused by transmission faults.
2. **Outlier Detection and Treatment:**
    - Descriptive statistical methods such as the **Interquartile Range (IQR)**.
    - Visualization tools like **boxplots** to highlight extreme values.

3. **Handling Missing Values:**
    - Removing instances with missing data (only when the loss is minimal).
    - Estimating missing values using methods like mean imputation or regression models.
    - Assigning probabilities to possible values in categorical attributes.
4. **Managing Duplicated and Inconsistent Data:**
    - Identifying duplicates by comparing multiple fields.
    - Standardizing data formats to avoid inconsistencies in merging datasets.

Data Cleaning and Preprocessing

The process of data cleaning ensures that machine learning models are trained on reliable and high-quality data. Key preprocessing tasks include:

- **Standardization:** Converting data into a common format.
- **Normalization:** Rescaling numerical values to a standard range.
- **Categorical Encoding:** Converting categorical values into numerical formats (e.g., one-hot encoding).

Impact of Data Quality on Machine Learning

High-quality data directly influences the performance and interpretability of machine learning models. Poor data quality can lead to:

- **Overfitting:** The model memorizes noise instead of learning generalizable patterns.
- **Misleading Insights:** Incorrect predictions that affect decision-making processes.
- **Increased Computational Costs:** Cleaning and preprocessing low-quality data require additional time and resources.

Ensuring data quality is a fundamental step in any data-driven project, significantly impacting the reliability of machine learning applications.

Association Rules

Introduction to Market Basket Analysis

Association rule mining is a fundamental technique in data mining aimed at discovering co-occurrences of items within transactions. It is widely used in market basket analysis to understand purchasing behaviors.

Objective:

- → Given a set of commercial transactions, find rules that predict the occurrence of an item based on the presence of other items in the same transaction.
- → Example of association rules:
  - {Diaper} → {Beer}
  - {Bread, Milk} → {Coke, Eggs}
  - {Beer, Bread} → {Milk}

Important Note:

- Implication in association rules refers to co-occurrence, not causality.
- Unlike logical (Boolean) implications, association rules hold true with a certain probability.

Frequent Itemsets and Rule Metrics

To identify useful patterns, key concepts include:

- **Itemset:** A collection of one or more items.
- **k-itemset:** An itemset containing k items.
- **Support Count (σ):** Frequency of occurrence of an itemset.
- **Support:** Fraction of transactions containing an itemset.
- **Frequent Itemset:** An itemset whose support meets or exceeds a minimum support threshold.

 Example Calculation:

- → Dataset of 5 transactions:
  - Support count of {Bread, Diaper, Milk} = 2
  - Support = 2/5 = 0.4

Definition of Association Rules

An association rule has the form:

A → C

- A: Antecedent (LHS - Left Hand Side)
- C: Consequent (RHS - Right Hand Side)

Example:

- {Diaper, Milk} → {Beer}

Rule Evaluation Metrics:

- **Support:** Fraction of transactions that contain both antecedent and consequent.
- **Confidence:** Likelihood that transactions containing the antecedent also contain the consequent.

Example Calculation:

- Support({Beer, Diaper, Milk}) = 2/5 = 0.4
- Confidence({Beer, Diaper} → {Milk}) = Support({Beer, Diaper, Milk}) / Support({Beer, Diaper})

Support and Confidence Considerations

- Rules with **low support** may be generated by random associations.
- Rules with **low confidence** may not be reliable.
- However, a rule with **low support but high confidence** can indicate an uncommon but important pattern.

Association Rule Mining Task

Given a set of transactions **N**, the goal is to find rules satisfying:

- **Support ≥ minsup (minimum support threshold)**
- **Confidence ≥ minconf (minimum confidence threshold)**

Brute-force Approach (Computationally Expensive):

1. Generate all possible rules.
2. Compute support and confidence for each rule.
3. Discard rules that fail to meet the thresholds.

Optimized Two-Step Approach:

1. **Frequent Itemset Generation:** Identify all itemsets meeting the minimum support threshold.
2. **Rule Generation:** Generate high-confidence rules from frequent itemsets by partitioning them.

Frequent itemset generation is the most computationally expensive step, requiring efficient algorithms like **Apriori**.

Frequent Itemset Generation

Generating frequent itemsets efficiently is a key challenge in association rule mining. The naive approach of listing all possible itemsets and checking their support is computationally prohibitive. Instead, efficient methods are required to prune unnecessary computations.

Brute-Force Approach

1. Generate all possible itemsets from the dataset (complexity: $O(2^D)$ for **D** items).
2. Scan the database to count occurrences of each itemset.
3. Remove itemsets that do not meet the **minimum support** threshold.
4. Computationally expensive and impractical for large datasets.

The Apriori Principle

If an itemset is **frequent**, then all its subsets must also be **frequent**.

Conversely, if an itemset is **not frequent**, then none of its supersets can be frequent.

This principle allows pruning large numbers of infrequent itemsets early in the process.

The Apriori Algorithm

1. **Candidate Generation (Join Step):**
    - Generate candidate **k+1**-itemsets by joining **k**-itemsets.
2. **Pruning (Prune Step):**
    - Remove any candidate **k+1**-itemset if any of its **k**-item subsets is not frequent.
3. **Support Counting:**
    - Scan transactions to compute support for candidates.
4. **Repeat Until Termination:**
    - Continue until no more frequent itemsets are found.

Example of Apriori Execution

1. Start with **L1**: frequent single items.
2. Generate **C2**: candidate pairs.
3. Compute support for **C2**, discard those below minsup.
4. Generate **C3** using surviving **L2** itemsets.
5. Continue until no further frequent itemsets can be generated.

Factors Affecting Complexity

- **Minimum Support Threshold:** Lower thresholds lead to more frequent itemsets, increasing computational cost.
- **Dimensionality of the Dataset:** More items mean higher complexity.
- **Database Size:** Larger databases require more scans, increasing runtime.
- **Transaction Width:** Longer transactions increase the number of subsets to evaluate.

Efficient frequent itemset generation is crucial for association rule mining, and the Apriori algorithm remains one of the most widely used approaches due to its pruning strategy and scalability.

Multidimensional Association Rules

Multidimensional association rules extend standard association rules by incorporating multiple attributes beyond item occurrences. These rules

consider attributes such as customer demographics, transaction times, and other external factors.

Comparison: Mono vs. Multidimensional Rules

- → **Mono-dimensional (intra-attribute):** The rule involves only one attribute (e.g., items purchased together in a transaction).
- → **Multidimensional (inter-attribute):** The rule considers multiple attributes (e.g., customer age, purchase category, and item bought together).

Equivalence Between Mono and Multi-dimensional Rules

Multidimensional rules can often be transformed into mono-dimensional ones by reformatting attributes as categorical items (e.g., transforming "Customer Age: 30-40" into an item category).

Handling Quantitative Attributes

- → Many association rule mining techniques focus on categorical data.
- → Quantitative attributes require **discretization**, such as:
  - Equi Frequency binning.
  - Clustering-based discretization.

Multilevel Association Rules

Multilevel association rules introduce hierarchical relationships within the dataset. These rules allow mining at different levels of abstraction.

Concept Hierarchy in Association Rules

- ✓ **Example:** Instead of just mining rules at the product level, rules can be extracted at broader levels such as product categories (e.g., "Fruits" instead of just "Apple").
- ✓ **Example Rule at Different Levels:**
  - **Specific level:** {Apple} → {Milk}
  - **Generalized level:** {Fruit} → {Dairy}.

Impact on Support and Confidence

- → **From Specialized to General:**

    - As abstraction increases, rule **support** generally increases (e.g., {Fruit} → {Dairy} may be more frequent than {Apple} → {Milk}).
    - More general rules might become interesting due to their broader applicability.
- → **From General to Specialized:**
    - As specificity increases, rule **support** may decrease.
    - Some highly specific rules may fall below the minimum support threshold.

Mining Multilevel Association Rules

- Rules are extracted at each hierarchical level in a **top-down** fashion.
- **Lower levels require lower support thresholds** because specific item combinations occur less frequently.

Conclusion on Association Rules

- **Mono-dimensional rules** focus on a single transaction attribute, while **multidimensional rules** incorporate additional data dimensions.
- **Multilevel rules** introduce hierarchical abstraction, balancing between generalization and specificity.
- Effective association rule mining requires **efficient frequent itemset discovery** and **rule pruning techniques** to ensure meaningful insights.

Classification

Introduction to Classification

Classification is a fundamental supervised learning technique where a model learns to assign labels to new data points based on training examples.

Supervised vs. Unsupervised Classification

- **Supervised Classification (Standard Classification):**
  - Each data point in the training set has a known label.
  - The model learns from labeled examples and predicts the class of new instances.
- **Unsupervised Classification:**
  - Also called clustering, but distinct from classification.
  - Groups similar instances without predefined labels.
  - Not covered in this course as “classification” always refers to supervised learning.

Key Elements of Supervised Classification

- **Dataset X**: Contains N instances, each with D attributes.
- **Class Vector Y**: Contains the class labels for each instance.
- **Class Labels Cc**: A finite set of possible values (e.g., disease types in medical diagnosis).
- **Supervision**: Class labels are assigned by domain experts.

Objective of Classification

The goal of classification is to learn a model that can predict the class label of new, unseen instances based on past labeled examples.

Classification Model

A classification model is an algorithm that, given a new instance, assigns it to one of the predefined classes.

Building a Classification Model

1. **Choosing a Learning Algorithm:** Select an appropriate classification technique.
2. **Training the Model:** Optimize parameters based on a labeled dataset.
3. **Evaluating Performance:** Assess the model's accuracy on test data.

Mathematical Representation

A classifier is a decision function: $M(x, \theta) = ypred$ where:

- x is the input data.
- heta heta represents model parameters.
- ypred is the predicted class label.

The learning process involves adjusting θ to minimize the prediction error.

Vapnik-Chervonenkis (VC) Dimension

- Measures a model's ability to separate data into distinct classes.
- A model with VC dimension N can perfectly classify any dataset with NN points.
- Example: A straight line has a VC dimension of 3.

Workflow for Classification

1. **Learning the Model:**
    - Train the model on a labeled dataset.
    - Ensure the training set is representative (random sampling is recommended).
    - Fit parameters using learning algorithms.
2. **Evaluating the Model:**
    - Use a separate test set with known labels.
    - Compare predicted labels with actual labels to measure accuracy.
3. **Applying the Model to New Data:**
    - The trained model classifies new, unseen instances.
    - Real-world deployment may involve continuous updates and accuracy monitoring.

Types of Classification Approaches

→ **Crisp Classification:** Assigns each instance a single label.

→ **Probabilistic Classification:** Outputs probability distributions over all possible labels.

Classification with Decision Trees

Introduction to Decision Trees

Decision trees are a widely used classification technique due to their interpretability and efficiency. They provide a flowchart-like model where decisions are made at internal nodes, leading to final class labels at leaf nodes.

Historical Background

- **ID3 (1966):** First decision tree model.
- **CLS (1979):** A refinement of ID3.
- **C4.5 (1993):** A widely used version developed by Quinlan.

Using a Decision Tree for Classification

A decision tree consists of:

- **Inner Nodes:** Represent attribute-based tests.
- **Leaf Nodes:** Represent class labels.

The classification process follows the tree structure, applying tests at each node to determine the path to a final prediction.

Decision Tree Construction

A decision tree is built recursively using the following process:

1. If all training examples belong to a single class, create a leaf node with that class label.
2. Otherwise, select the best attribute to split the data.
3. Create child nodes corresponding to the possible values of the attribute.
4. Recursively apply the process to each child node until stopping criteria are met (e.g., all nodes are pure or data size is too small).

Entropy and Information Gain

Concept of Entropy in Classification

Entropy is a fundamental concept in information theory that quantifies the uncertainty in a dataset. In classification, entropy is used to measure the impurity of a dataset regarding class labels.

Definition of Entropy

For a dataset X with class labels C, entropy is given by:

$$H(C) = -\sum_{i=1}^{|C|} p(c_i) \, log_2 p(c_i)$$

where:

- $p(c_i)$ is the probability of class cic_i in the dataset.
- Entropy is maximized when all classes have equal probability.
- Entropy is 0 when all samples belong to a single class.

Information Gain (IG)

Information gain measures the reduction in entropy after splitting the dataset based on an attribute. The goal is to choose the attribute that maximizes information gain.

$$IG(C|A) = H(C) - H(C|A)$$

where:

- H(C)is the entropy of the entire dataset.
- H(C∣A) is the weighted sum of entropies after splitting on attribute A.

Choosing the Best Attribute

The attribute with the highest information gain is selected as the best splitting criterion. The process is repeated recursively to grow the decision tree.

Example: Splitting on Petal Width in the Iris Dataset

- → Before the split: H(C)= 1
- → After splitting at petal width=1.75 : H(C ∣ petal width)=0.31
- → IG(C ∣ petal width)= 1 − 0.31 = 0.69I

A higher IG indicates a better attribute for classification.

Learning a Decision Tree

Building a Decision Tree

A decision tree is constructed recursively by selecting the best attribute at each step to split the dataset and maximize information gain.

Steps to Learn a Decision Tree

1. **Choose the Best Attribute:**
   - Use Information Gain (IG) or other impurity measures to find the attribute that best separates the data.
2. **Split the Dataset:**
   - Divide the dataset based on the chosen attribute's values or a threshold for continuous attributes.
3. **Recursion on Subsets:**
   - Repeat steps 1 and 2 for each subset until a stopping criterion is met (e.g., all instances belong to the same class, or data size is too small).

✓ Example: Learning a Decision Tree from the Iris Dataset

1. The best attribute to split first is **Petal Width** (threshold: 0.75 cm).
2. Instances are divided into two subsets:
   - → **Left branch:** Petal width < 0.75 → all belong to class 0 (Setosa).
   - → **Right branch:** Further splits occur based on Petal Length and Sepal Length.

Handling Continuous Attributes

- Continuous attributes require threshold-based splits (e.g., x < 1.75x < 1.75).
- The optimal threshold is determined by maximizing information gain.

Stopping Criteria

A decision tree stops growing when:

- ✓ All instances in a node belong to the same class.
- ✓ No further attribute provides a significant information gain.
- ✓ The dataset becomes too small to split further.

Errors and Overfitting

Training and Test Errors

Errors in decision trees can occur due to incorrect splits, noise in data, or model complexity. There are two types of errors to consider:

- → **Training Set Error:** The proportion of misclassified instances within the training data.
- → **Test Set Error:** The proportion of misclassified instances when applying the model to unseen data.

Why Test Error is Higher than Training Error

In real-world applications, the test error is typically higher than the training error because the model may have learned noise or specific patterns from the training set that do not generalize to new data.

Overfitting in Decision Trees

Definition: Overfitting occurs when a model learns the training data too well, including noise, and fails to generalize to unseen data.

Indicators of Overfitting

- ✘ **High accuracy on training data, but poor performance on test data.**
- ✘ **Excessively complex trees** with many branches capturing noise rather than true patterns.
- ✘ **Sharp differences between training and test set errors.**

Causes of Overfitting

1. **Presence of Noise:**
   Some training instances may contain errors, leading the model to learn incorrect patterns.
2. **Lack of Representative Data:**
   If some real-world situations are underrepresented in training data, the model might not generalize well.

Reducing Overfitting: Pruning Decision Trees

Occam's Razor Principle:

Everything should be made as simple as possible, but not simpler.

Pruning helps simplify trees by removing unnecessary branches.
Methods for pruning include:

- **Pre-Pruning:** Stop growing the tree early, based on threshold criteria.
- **Post-Pruning:** Grow the full tree and then remove branches that do not improve performance.

Impurity Functions

Measuring Node Impurity

Impurity functions are used in decision trees to evaluate how mixed a node is regarding class labels. A pure node contains instances from only one class, while an impure node contains a mix of classes.

Common Impurity Functions

1. **Entropy:**

   - Measures the level of uncertainty in class distribution.
     Formula:

$$H(C) = -\sum_{i=1}^{|C|} p(c_i)\, log_2 p(c_i)$$

   - Used in algorithms like ID3 and C4.5.

2. **Gini Index:**

    - Measures impurity based on class probabilities.
      Formula:

$$GINI(C) = 1 - \sum_{i=1}^{|C|} p(c_i)^2$$

    - Lower values indicate *purer nodes*.
    - Used in CART (Classification and Regression Trees).

3. **Misclassification Error (Optional):**

    - Measures the fraction of incorrectly classified instances.
      Formula:

$$ME(C) = 1 - p(c_i)$$

    - Less sensitive to class distributions compared to entropy and Gini.

Comparison of Impurity Functions

- **Entropy and Gini Index** behave similarly but Gini is computationally simpler.
- **Misclassification Error** is less effective as a splitting criterion due to its linear nature.
- Decision trees typically use **Entropy (C4.5)** or **Gini (CART)** for splitting nodes.

Final Remarks on Decision Trees

Complexity of Decision Tree Induction

Decision tree induction is computationally efficient but can become complex depending on dataset characteristics. The main factors affecting complexity include:

- **Number of instances (N)**: More data increases processing time.
- **Number of attributes (D)**: More features require more comparisons.
- **Tree depth (h)**: Determines the number of decisions required for classification.
- **Splitting method**: Binary splits are computationally simpler than multi-way splits.

The overall time complexity of decision tree construction is $O(DN\,log(N))$, making it feasible for large datasets.

Characteristics of Decision Trees

1. **Non-Parametric Learning:**
    - No assumptions on class distribution or attribute relationships.
2. **Efficient Classification:**
    - Once built, classification has complexity **O(h)** (where h is tree height).
3. **Robustness to Noise:**
    - Overfitting can be controlled through pruning.
4. **Handling Redundant Attributes:**
    - Trees automatically ignore uninformative attributes.
5. **Handling Correlated Attributes:**
    - If one attribute is chosen for a split, the correlated ones may be ignored.

Impact of Hyperparameters

- **Tree Depth:**
    - Shallow trees generalize better but may underfit.
    - Deep trees capture details but risk overfitting.
- **Pruning Strategy:**
    - Pre-pruning limits tree growth early.
    - Post-pruning removes unnecessary branches after full growth.
- **Impurity Measure:**
    - Entropy vs. Gini Index has minimal impact on accuracy.
- **Splitting Strategy:**
    - Multi-way vs. binary splits affect interpretability and performance.

Conclusion

Decision trees are one of the most interpretable and widely used machine learning models. They serve as a solid baseline for classification tasks due to their simplicity, efficiency, and ability to handle both continuous and categorical data.

Evaluation of a Classifier

Why Evaluate a Classifier?

The evaluation of a classifier is crucial to determining its effectiveness and guiding the choice of the best model. Key questions to address include:

- ✓ Which classification model performs best?
- ✓ Which algorithm is most effective?
- ✓ What is the optimal parameter configuration?

The Oil Slick Example

A practical case of classifier evaluation involves detecting oil slicks from satellite images:

- Oil spills appear as dark regions in radar images, but similar patterns can be caused by weather conditions.
- Manual detection is costly and time-consuming.
- An automatic hazard detection system helps pre-select images for human review.
- The challenge is balancing **false alarms** and **undetected spills** using performance evaluation.

Training, Validation, and Test Sets

- → **Training Set:** Used to build the model.
- → **Validation Set (optional):** Used for tuning parameters.
- → **Test Set:** Provides an independent performance estimate.

Using more training data generally improves model performance, but we must balance training and evaluation resources.

Empirical Error Estimation

- The test error provides an estimate of real-world classification performance.
- The test set should be representative of the overall dataset.
- Error estimation involves probabilistic variability due to random variations in training and test data.

Confidence Intervals in Error Estimation

- Predictions on the test set follow a Bernoulli process (success = correct classification, failure = error).
- The empirical error rate is an estimate of the true error probability.
- Confidence intervals provide a range within which the true error rate is likely to fall.

Wilson Score Interval for Error Estimation

- Used to calculate a confidence range for classification error rates.
- The interval width depends on the number of test samples and the chosen confidence level.
- Increasing the test set size narrows the uncertainty range.

Statistical Pruning of Decision Trees

- C4.5 prunes subtrees based on error estimation.
- A subtree is replaced by a leaf if:
    - The estimated error at the root of the subtree is lower than the weighted sum of leaf errors.
    - This tradeoff reduces complexity while maintaining accuracy.

Model Selection for a Classifier

Why Model Selection Matters

Model selection aims to determine the best classifier and its optimal parameter configuration. A well-selected model improves accuracy, efficiency, and generalization to new data.

Train, Test, Evaluate, Optimize

A systematic approach to selecting and fine-tuning models:

1. **Train the Model:** Fit the model to the training data.
2. **Evaluate Performance:** Test on validation or test data to measure accuracy.
3. **Optimize Hyperparameters:** Adjust parameters to maximize performance.
4. **Final Testing:** Assess the final model on independent test data.

Train/Test Split Strategies

- ✓ **Small Datasets:** Allocate more data for training (e.g., 80/20 or 90/10 split).
- ✓ **Large Datasets:** A 70/30 split allows for sufficient testing.
- ✓ **Simple Models:** Require less training data, allowing for a larger test set.
- ✓ **Complex Models:** Require more training data (e.g., deep learning models may need 90/10 or 85/15 splits).

Cross-Validation: A Robust Approach

Cross-validation improves evaluation reliability by repeatedly splitting the dataset:

- ✓ **k-Fold Cross-Validation:** The dataset is divided into kk subsets, with each subset serving as a test set once while the others are used for training.
- ✓ **Bootstrap Sampling:** Randomly selects training samples with replacement while using unselected instances for testing.

Hyperparameter Optimization

Optimizing model hyperparameters is crucial for improving performance:

- ✓ **Grid Search:** Exhaustively searches over predefined hyperparameter values.
- ✓ **Randomized Search:** Randomly samples a subset of hyperparameter values.

✓ **Bayesian Optimization:** Uses probabilistic models to find optimal configurations efficiently.

Final Model Selection Process

1. **Select Candidate Models:** Compare classifiers such as Decision Trees, SVMs, and Neural Networks.
2. **Optimize Hyperparameters:** Use cross-validation to find the best configuration.
3. **Evaluate the Best Model:** Perform a final assessment on the test set.
4. **Deploy and Monitor:** Implement the model and track real-world performance.

Performance Measures of a Classifier

Beyond Accuracy: Evaluating Classifier Performance

Accuracy alone is not always a reliable performance metric, especially for imbalanced datasets. Additional evaluation metrics help assess different aspects of classification quality.

Confusion Matrix

A confusion matrix summarizes classification results by comparing predicted and actual class labels:

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

From the confusion matrix, various performance metrics are derived.

Key Performance Metrics

1. **Precision (Positive Predictive Value):**
   Measures how many predicted positives are actually correct.

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

2. **Recall (Sensitivity, True Positive Rate):**
   Measures how many actual positives are correctly identified.

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

3. **Specificity (True Negative Rate):**
   Measures how well negatives are classified.

$$Specificity = \frac{TN}{TN+FP}$$

4. **F1-Score:**
   Harmonic mean of Precision and Recall. It's useful when balancing Precision and Recall is crucial.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Multi-Class Evaluation

- **Macro Average:** Averages the metric across all classes equally.
- **Weighted Average:** Averages the metric while considering class proportions.

Beyond Counting: Considering Chance and Cost

→ **Cohen's Kappa (κ):** Measures agreement between predicted and true labels while accounting for random chance.

- **Matthews Correlation Coefficient (MCC):** A balanced measure even for imbalanced datasets.
- **Cost-Sensitive Learning:** Weighs classification errors differently based on application needs (e.g., medical diagnostics).

Evaluation of a Probabilistic Classifier

Probabilistic vs. Crisp Classification

Many classifiers output probability distributions instead of hard class labels. This is useful when:

- A classification decision depends on multiple factors.
- Further evaluation steps refine the decision-making process.

Converting Probabilities to Class Labels

- ✓ **Binary Classification:**
  - A threshold is set (e.g., 0.5) to decide the positive class.
- ✓ **Multi-Class Classification:**
  - The class with the highest probability is assigned.

Lift Chart

- Evaluates classifier performance by ranking predictions by probability.
- Compares the proportion of true positives in the highest probability samples vs. a random selection.
- The greater the gap between the model and random chance, the better the classifier.

ROC Curve (Receiver Operating Characteristic)

Originated from radar signal analysis. It plots **True Positive Rate (TPR) vs. False Positive Rate (FPR)** at various thresholds.

The area under the ROC curve (AUC) measures classification performance:

- **AUC = 1.0:** Perfect classifier.
- **AUC = 0.5:** No discrimination (random classifier).
- **Higher AUC** indicates better model performance.

Choosing the Best Threshold

- ROC analysis helps select the threshold that balances sensitivity (recall) and specificity based on application needs.
- A medical diagnosis model might prefer a threshold that minimizes false negatives, while fraud detection may focus on reducing false positives.

Statistical Modeling – Naive Bayes Classifier

Overview

The **Naive Bayes Classifier** is a probabilistic classification technique based on **Bayes' Theorem**. It assumes that attributes are **conditionally independent** given the class. This assumption is strong and often unrealistic, yet the method performs well in practice.

Main Characteristics

- **Based on statistics**, particularly **Bayes' Theorem**.
- **Considers all attributes** while classifying.
- **Assumes conditional independence** of attributes given the class.
- **Estimates probabilities using frequencies** from the training data.

Bayes' Theorem

Given a hypothesis HH and an evidence E, Bayes' theorem states:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

For classification:

- H represents a class.
- E represents the feature values of the instance to be classified.
- Assuming attribute independence:

$$P(C|E) = \frac{P(E_1|C)P(E_2|C)...P(E_n|C)P(C)}{P(E)}$$

Since P(E) is constant for all classes, it can be omitted in classification decisions.

A Fictitious Example: Weather and Play Dataset

Consider the dataset where **Play (Yes/No)** depends on four attributes: **Outlook, Temperature, Humidity, and Windy**. The probability of a new instance is calculated using attribute frequencies.

✓ **Example:**

- Outlook = Sunny
- Temperature = Cool
- Humidity = High
- Windy = True

Using the dataset's probabilities, we compute the likelihoods:

$$P(\text{Yes}) = (2/9)\times(3/9)\times(3/9)\times(3/9)\times(9/14) = 0.0053$$

$$P(\text{No}) = (3/5)\times(1/5)\times(4/5)\times(3/5)\times(5/14) = 0.0206$$

Normalizing:

$$P(\text{Yes}) = \frac{0.0053}{0.0053+0.0206} = 20.5\%$$

$$P(\text{No}) = \frac{0.0206}{0.0053+0.0206} = 79.5\%$$

Since **P(No) > P(Yes)**, the classifier predicts **No**.

Handling Missing Values

- **Test instances:** If an attribute is missing, its probability is omitted.
- **Training instances:** Instances with missing values are ignored when computing frequency-based probabilities.

Handling Numeric Values

The frequency-based method does **not** work directly for numerical values.
A **Gaussian distribution** is assumed for numerical attributes.

The probability density function is used:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Where:

- μ = mean of the attribute within a class.
- σ = standard deviation.

For a new instance, plug in the attribute value, mean, and standard deviation to obtain likelihoods.

Example: If Temperature = 66 and μ=73, σ=6.2,

$$f(66|Yes) = \frac{1}{\sqrt{2\pi 6.2}} e^{-\frac{(66-73)^2}{2*6.2^2}} = 0.0340$$

Then, the likelihoods for **Yes** and **No** are computed similarly to categorical cases.

Laplace Smoothing

When an attribute value never appears in a class, its probability becomes **zero**, which is problematic.

Laplace smoothing prevents this by adding a smoothing factor α\alpha:

$$P(d = v|C) = \frac{count(d=v,C)+\alpha}{count(C)+\alpha V}$$

Where:

- V = number of distinct values for the attribute.
- α is typically set to **1**.

Summary

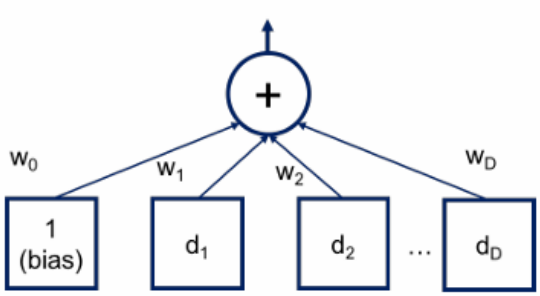
- Naive Bayes provides a **clear probabilistic model**.
- It is **effective despite the independence assumption**.
- **Fails when independence is violated**, especially if attributes are correlated.
- **Works best with categorical data** but can handle numerical values with Gaussian assumptions.
- **Laplace smoothing** helps when attribute values are missing for a class.

Linear Classification with the Perceptron

Overview

The **Perceptron** is one of the simplest linear classifiers. It was introduced by **Frank Rosenblatt** in 1958 and is often referred to as an **artificial neuron**. The key idea is to find a **hyperplane** that separates data points of different classes.

Main Characteristics



- **Supervised learning algorithm** for binary classification.
- **Linear model**, meaning it works best with linearly separable data.
- **Iterative training process** that updates weights based on misclassified points.
- **Converges only if the data are linearly separable**.

The Perceptron Model

A perceptron computes a **weighted sum** of its inputs:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D$$

where:

- x_i are the input features.
- w_i are the weights.
- w_0 is the bias.

The **activation function** is a **step function**:

$$\hat{y} = \begin{cases} +1, & \text{if } y > 0 \\ -1, & \text{if } y \leq 0 \end{cases}$$

This function determines whether an input belongs to **class +1 or -1**.

Learning the Hyperplane

The perceptron **learns** by iteratively adjusting the weights to reduce misclassification. The goal is to find a **hyperplane** such that:

→ All **positive examples** are on one side.
→ All **negative examples** are on the other side.

Perceptron Learning Algorithm

1. **Initialize** all weights to zero.
2. **Repeat** until all points are correctly classified (or max iterations reached):
   → For each training instance:
      ✓ Compute $y = w_0 + w_1 x_1 + \ldots + w_D x_D$
      ✓ If correctly classified → **do nothing**.
      ✓ If misclassified → **update weights**:

$$w_i = w_i + \eta y_{true} x_i$$

Perceptron Convergence

- The perceptron **converges** if the dataset is **linearly separable**.
- If the dataset is **not linearly separable**, the algorithm will **never converge**.

- A **maximum number of iterations** is used to stop training.

Limitations of the Perceptron

1. **Requires linear separability** – If data is not linearly separable, it does not converge.
2. **No probability outputs** – Unlike Naive Bayes, it does not provide a probability score.
3. **No handling of non-linearity** – Cannot learn complex decision boundaries.

Summary

- The **Perceptron** is a **linear classifier** that finds a hyperplane separating two classes.
- It **updates weights iteratively** based on misclassified points.
- **Only converges for linearly separable data**.
- **Fails on non-linearly separable data** and requires additional modifications.

Support Vector Machines (SVM)

Overview

Support Vector Machines (SVM) are powerful supervised learning models used for **binary classification**. They aim to find the best decision boundary, called the **maximum margin hyperplane**, that separates data points belonging to different classes.

Main Characteristics

- ✓ **Effective for high-dimensional spaces**.
- ✓ **Can model non-linearly separable data using kernels**.
- ✓ **Robust to overfitting**, especially with a proper choice of the regularization parameter.
- ✓ **Uses support vectors** to define the decision boundary.

Maximum Margin Hyperplane

For a **linearly separable** dataset, SVM finds the hyperplane that maximizes the margin, i.e., the distance between the hyperplane and the nearest data points (support vectors).

Mathematically, it solves the optimization problem:

$$\max_{w_0, w_1, \ldots, w_D} M$$

subject to:

$$\sum_{j=1}^{D} w_j^2 = 1$$

$$c_i(w_0 + w_1 x_{i1} + \ldots + w_D x_{iD}) \geq M, \forall i$$

where M is the margin and C_i represents class labels (-1 or 1).

Soft Margin and Non-Separable Data

- In **real-world scenarios**, data is often **not perfectly separable**.
- A **soft margin** approach allows some misclassification by introducing a penalty parameter **C** that balances margin width and classification error.
- A large **C** leads to a smaller margin with fewer misclassified points, while a small **C** results in a larger margin but more misclassified points.

Handling Non-Linear Data: The Kernel Trick

- SVM can handle **non-linearly separable data** by transforming it into a higher-dimensional space where a linear decision boundary can be applied.
- Instead of explicitly computing the transformation, **kernel functions** are used to compute the dot product in the transformed space efficiently.

Common Kernels

1. **Linear Kernel**: $K(x, x') = x \cdot x'$
2. **Polynomial Kernel**: $K(x, x') = (\gamma x \cdot x' + r)^d$
3. **Radial Basis Function (RBF) Kernel**: $K(x, x') = e^{-\gamma ||x - x'||^2}$
4. **Sigmoid Kernel**: $K(x, x') = tanh(\gamma x \cdot x' + r)$

Complexity

The time complexity of SVM depends on the optimization library used.

- **LibSVM**, a popular SVM library, has a complexity that scales from **O(D * N²)** to **O(D * N³)**, depending on data sparsity.

Advantages of SVM

✓ **Effective for high-dimensional data**.
✓ **Works well even with small datasets**.
✓ **Uses support vectors**, reducing the impact of irrelevant data points.
✓ **Not affected by local minima**, unlike neural networks.

Disadvantages of SVM

- ✘ **Slow training for large datasets**.
- ✘ **Choice of kernel and parameters requires tuning**.
- ✘ **Does not directly provide probability estimates**, but probability scores can be computed separately.

Summary

- → **SVM aims to maximize the margin** between two classes.
- → **Uses support vectors** to define the decision boundary.
- → **Soft margin SVM** allows handling non-separable data.
- → **Kernels enable non-linear classification** by mapping data to higher dimensions.
- → **Regularization parameter C** controls the trade-off between margin size and misclassification.

Neural Networks

Overview

Neural Networks (NN) are computational models inspired by biological neural networks. They are designed to recognize patterns and are particularly effective for **non-linear classification problems**.

Main Characteristics

- **Composed of multiple layers of interconnected neurons**.
- **Capable of learning complex decision boundaries**.
- **Uses activation functions to introduce non-linearity**.
- **Trained using backpropagation and gradient descent**.

Multi-Layer Perceptron (MLP)

A **multi-layer perceptron (MLP)** is a type of neural network that consists of multiple layers:

1. **Input Layer**: Represents the input features.

2. **Hidden Layer(s)**: Process information using weights and activation functions.
3. **Output Layer**: Produces the final classification or regression output.

Activation Functions

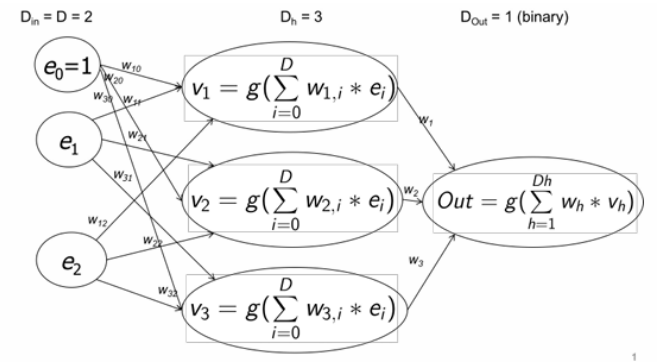
Activation functions introduce non-linearity into the network, allowing it to learn more complex patterns.

Common activation functions:

- **Sigmoid**: $f(x) = \frac{1}{1+e^{-x}}$ (squashes input between 0 and 1)
- **ReLU (Rectified Linear Unit)**: $f(x) = max(0, x)$ (avoids vanishing gradient problem)
- **Tanh**: $f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ (maps input between -1 and 1)

Feedforward Neural Networks

- **Each layer passes information forward to the next layer**.
- **Weights are adjusted through backpropagation** to minimize classification error.



Training Neural Networks

The **training process** involves adjusting the network's weights to minimize the error:

1. **Forward Propagation**: Inputs pass through layers to produce an output.
2. **Compute Error**: The difference between predicted and actual values.

3. **Backpropagation**: The error is propagated backward to adjust weights.
4. **Gradient Descent**: Optimization algorithm to minimize the error.

Gradient Descent and Learning Rate

- **Gradient Descent** finds the optimal weights by minimizing the loss function.
- The **learning rate** controls the step size of weight updates.
- Variants include **stochastic gradient descent (SGD)** and **batch gradient descent**.

Training Algorithms

- **Stochastic Gradient Descent (SGD)**: Updates weights after each training instance.
- **Batch Gradient Descent**: Updates weights after processing an entire batch.
- **Mini-Batch Gradient Descent**: Updates weights after processing small batches.

Computing Weight Corrections

The **error function** is typically measured as the sum of squared errors:

$$E(w) = \frac{1}{2}(y - Transfer(w, x))^2$$

where **Transfer** represents the activation function applied to a node.

- **Convex vs. Non-Convex Error Functions**: Some loss functions have multiple local minima, making optimization challenging.
- **Gradient-Based Updates**: Weight updates follow the negative gradient direction to minimize error.

Backpropagation Algorithm

1. **Compute the error at the output layer**.
2. **Propagate the error backward to compute weight changes**.
3. **Update weights in hidden layers** accordingly.
4. **Repeat for multiple epochs until convergence is reached**.

Learning Modes

A. **Stochastic Learning**: Updates weights after each training sample (introduces noise but prevents local minima stagnation).
B. **Batch Learning**: Updates weights after processing an entire dataset (more stable but computationally expensive).

Overfitting and Regularization

- **Overfitting** occurs when the model memorizes training data but fails on new data.
- **Regularization techniques**:
  - **Dropout**: Randomly deactivates neurons during training.
  - **L2 Regularization (Weight Decay)**: Adds a penalty term to large weights.
  - **Early Stopping**: Stops training when validation error stops improving.

Neural Network Architectures

- **Feedforward Neural Network (FNN)**: The simplest type, where information flows in one direction.
- **Convolutional Neural Network (CNN)**: Designed for image processing.
- **Recurrent Neural Network (RNN)**: Used for sequential data like text and time series.
- **Transformer Models**: Used for NLP applications (e.g., BERT, GPT).

Example: Character Recognition

- **Input**: A 7x5 matrix representing characters with noise.
- **Hidden Layer**: Up to 60 nodes processing intermediate features.
- **Output Layer**: 26 nodes, each representing a letter.
- **Error Analysis**: Performance evaluated by varying noise and network complexity.

Advantages of Neural Networks

✓ Handles complex and non-linear relationships.
✓ Can learn hierarchical feature representations.
✓ Scalable with large amounts of data.

Disadvantages of Neural Networks

- ✘ Requires significant computational power.
- ✘ Prone to overfitting if not regularized.
- ✘ Hard to interpret compared to simpler models.

Summary

- **Neural Networks** are powerful models for complex classification and regression tasks.
- MLPs consist of input, hidden, and output layers.
- Trained using backpropagation and gradient descent.
- Overfitting can be mitigated using dropout, L2 regularization, and early stopping.
- Different architectures exist, such as CNNs, RNNs, and Transformers.

Instance Learning - K Nearest Neighbours Classifier

Overview

The **K-Nearest Neighbours (KNN) Classifier** is an **instance-based learning** algorithm, meaning that it does not explicitly construct a model but instead *relies on storing the entire training dataset*. When a new instance needs to be classified, KNN finds the **K closest points** in the training set and assigns the most common class among them.

Main Characteristics

- **Lazy Learning**: The algorithm does not perform explicit training; it simply stores training data and classifies new instances by comparison.
- **Instance-Based**: Instead of learning an explicit model, KNN directly uses the training samples.
- **Non-Parametric**: KNN does not assume any prior distribution of the data.
- **Highly Dependent on Distance Metric**: The choice of distance function significantly affects the classifier's performance.

Algorithm Steps

1. Choose the number of neighbors **K**.
2. Compute the distance between the test instance and all training instances.
3. Select the **K** closest instances.
4. Assign the most frequent class label among the K neighbors.

Distance Metrics

KNN relies on measuring similarity between data points using different distance functions. Common choices include:

a. **Euclidean Distance**:

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

b. **Manhattan Distance**:

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

c. **Minkowski Distance** (generalization of Euclidean and Manhattan):

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

Setting p=1 gives Manhattan distance, and p=2 gives Euclidean distance.

d. **Mahalanobis Distance** (takes into account correlation between features):

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

where S is the covariance matrix of the dataset.

Choosing the Right Value of K

- **Small K**: The model is sensitive to noise and may lead to overfitting.
- **Large K**: The model becomes more stable but loses fine-grained details, possibly underfitting.
- **Common Choices**: Odd values of **K** (e.g., 3, 5, 7) are often used to avoid ties in binary classification.

Weighting the Neighbors

Instead of treating all **K** neighbors equally, some variations of KNN apply **weighting schemes** to give more importance to closer neighbors. A common approach is:

- **Inverse Distance Weighting**: $w_i = \frac{1}{d(x,x_i)}$

where w_i is the weight assigned to a neighbor and $d(x, x_i)$ is the distance between the test point and the neighbor.

Computational Complexity

- **Training Time Complexity**: O(1) (no actual training phase)
- **Prediction Time Complexity**: O(NlogN) (sorting the distances in large datasets)
- **Storage Complexity**: O(N), as the entire dataset is stored

Pros and Cons of KNN

Advantages	Disadvantages
✓ Simple and intuitive	✘ Computationally expensive for large datasets,
✓ Works well with well-structured data	✘ Sensitive to irrelevant features
✓ No training phase required	✘ Performance depends on distance metric and K value
✓ Effective in multi-class classification	✘ Memory-intensive, as it requires storing the entire dataset
✓ Flexible, can be used for both classification and regression	✘ Struggles with imbalanced datasets, where one class significantly outweighs others

Applications of KNN

- **Handwriting Recognition**: Used in Optical Character Recognition (OCR) systems.
- **Recommender Systems**: Identifying similar users/items based on past behavior.
- **Medical Diagnosis**: Predicting diseases based on patient symptoms.
- **Anomaly Detection**: Identifying fraudulent transactions or unusual network activity.

Summary

- **KNN is a non-parametric, instance-based classifier**.
- **Relies on distance metrics to find the closest neighbors**.
- **Can use weighted neighbors to improve performance**.
- **The choice of K significantly impacts performance**.
- **Computationally expensive for large datasets** but useful for many real-world applications.

Loss Function

Overview

A **loss function** quantifies how well a machine learning model is performing by measuring the difference between the predicted and actual values. The goal of training is to minimize this loss.

Common Types of Loss Functions

Regression Loss Functions

1. **Mean Squared Error (MSE)**: Penalizes larger errors more heavily.

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

2. **Mean Absolute Error (MAE)**: More robust to outliers than MSE.

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

3. **Huber Loss**: Combines MSE and MAE to be robust against outliers.

Classification Loss Functions

1. **Binary Cross-Entropy (Log Loss)**: Used for binary classification.

$$H(p, q) = -\sum y_i \log(\widehat{y_i}) + (1 + y_i)\log(1 - \widehat{y_i})$$

2. **Categorical Cross-Entropy**: Generalization of binary cross-entropy for multi-class problems.
3. **Hinge Loss**: Used for Support Vector Machines (SVM).

$$L = \sum max(0, 1 - y_i * \widehat{y_i})$$

Role of Loss Functions in Optimization

- Loss functions are minimized using optimization algorithms like **Gradient Descent**.
- A well-chosen loss function improves the model's performance and robustness.

Summary

- Loss functions measure the error in predictions.
- MSE and MAE are common in regression tasks.
- Cross-entropy and hinge loss are used in classification tasks.
- Minimizing the loss function is key to improving a model's accuracy.

From a Binary Classifier to Multi-Class Classification

Overview

Many classifiers, such as **Support Vector Machines (SVMs)** and **Perceptrons**, are inherently designed for **binary classification** (distinguishing between two classes). To extend them to handle multiple classes, we can use different strategies.

Strategies for Multi-Class Classification

There are two main approaches to extend binary classifiers to multi-class problems:

1. **One-vs-One (OvO) Strategy**
2. **One-vs-Rest (OvR) Strategy**

One-vs-One (OvO)

This approach trains a **separate binary classifier** for each **pair of classes**.

For **C** classes, this results in **C * (C - 1) / 2** classifiers.

During prediction, each classifier votes for a class, and the class with the most votes wins.

- **Advantages**:
  - ✓ Each classifier deals with a smaller subset of data, making training faster.
  - ✓ Works well with SVMs.
- **Disadvantages**:
  - ✘ Requires training many classifiers, making it computationally expensive.

One-vs-Rest (OvR)

This approach trains **C binary classifiers**, where each classifier distinguishes one class from all the others.

During prediction, the classifier with the highest confidence is chosen.

- **Advantages**:
  - ✓ Requires fewer classifiers than OvO (**C** classifiers instead of **C * (C - 1) / 2**).
  - ✓ Simpler and more interpretable.
- **Disadvantages**:
  - ✘ Can be **imbalanced**, as each classifier sees many negative examples but fewer positive examples.

When to Use OvO vs. OvR

- **OvO is better for algorithms like SVMs**, where training on large datasets is computationally expensive.
- **OvR is better for simpler models**, like logistic regression or naive Bayes, where training multiple classifiers is feasible.

Alternative Multi-Class Methods

Some classifiers natively support multi-class classification:

- **Decision Trees** (e.g., CART, ID3, C4.5)
- **Random Forests**
- **Neural Networks** (e.g., MLP, CNN, RNN)

Summary

- → Binary classifiers can be extended to multi-class problems using **One-vs-One (OvO)** or **One-vs-Rest (OvR)** strategies.
- → **OvO** trains multiple binary classifiers for **each class pair**, while **OvR** trains one classifier per class against all others.
- → Some classifiers natively support multi-class classification, avoiding the need for these strategies.

Ensemble Methods

Overview

Ensemble methods are techniques that **combine multiple classifiers** to improve prediction accuracy. The idea is that by aggregating multiple models, the overall performance is better than using a single model alone.

Why Use Ensemble Methods?

✓ Reduce **variance** (by averaging multiple models, e.g., bagging).
✓ Reduce **bias** (by correcting weak models, e.g., boosting).
✓ Improve **generalization** to unseen data.

Types of Ensemble Methods

1. Bagging (Bootstrap Aggregating)

- **Concept**: Train multiple classifiers on **randomly sampled subsets** of the dataset with replacement.
- **Final Prediction**: Take the majority vote (classification) or average prediction (regression).
- **Example Algorithm**: **Random Forest** (combination of decision trees trained with bagging).
- **Effect**: Reduces variance and helps prevent overfitting.
- **Key Parameters**:
    - **Number of base learners**: More learners increase stability.
    - **Sample size**: Fraction of dataset used per learner.

2. Boosting

- **Concept**: Sequentially trains weak classifiers, giving more weight to misclassified samples at each step.
- **Final Prediction**: Combines the weighted predictions of all classifiers.
- **Example Algorithms**:
    - **AdaBoost**: Assigns higher weights to misclassified examples.
    - **Gradient Boosting**: Sequentially fits new models to the residual errors of the previous models.
    - **XGBoost**: An optimized implementation of gradient boosting.

- **Effect**: Reduces bias and improves predictive accuracy.
- **Key Parameters**:
    - **Learning rate**: Determines the contribution of each model.
    - **Number of estimators**: More estimators improve accuracy but increase training time

3. Stacking (Stacked Generalization)

- **Concept**: Combines predictions from multiple base classifiers using a **meta-classifier**.
- **Final Prediction**: The meta-classifier learns how to best combine predictions.
- **Example**: Combining logistic regression, SVMs, and decision trees with a neural network as a meta-classifier.
- **Effect**: Can capture relationships not seen by individual models.
- **Key Parameters**:
    - **Choice of base models**: Diversity improves performance.
    - **Meta-learner selection**: A strong generalizer enhances stacking.

Comparison of Ensemble Methods

Method	Reduces Variance	Reduces Bias	Risk of Overfitting
Bagging	Yes	No	Low
Boosting	Yes	Yes	Higher
Stacking	Yes	Yes	Medium

Hybrid Approaches

Some methods **combine** ensemble techniques for better performance:

- **Bagging + Boosting**: Train base models with bagging and apply boosting.

- **Stacking + Bagging**: Use stacked classifiers as base models and apply bagging for stability.
- **Voting Classifiers**: Combine outputs of different models using majority voting or weighted voting.

Advantages of Ensemble Methods

✓ Improve accuracy over individual classifiers.
✓ More robust to overfitting (especially bagging).
✓ Can work with any type of base classifiers (decision trees, neural networks, SVMs, etc.).

Disadvantages of Ensemble Methods

✘ Increased computational complexity (training multiple models).
✘ Harder to interpret compared to a single model.
✘ Boosting can be sensitive to noisy data.

Summary

- **Bagging** reduces variance by averaging multiple models (**e.g., Random Forest**).
- **Boosting** improves weak classifiers by focusing on hard-to-classify examples (**e.g., AdaBoost, XGBoost**).
- **Stacking** learns to combine multiple models optimally using a meta-classifier.
- Ensemble methods improve performance but can increase computational cost.

Forest of Randomized Trees

Overview

A **Forest of Randomized Trees** is an ensemble method that constructs multiple decision trees and aggregates their predictions to improve accuracy and robustness. This method introduces randomness during tree construction to reduce overfitting and enhance generalization.

Key Characteristics

- **Uses multiple decision trees** instead of a single tree.
- **Introduces randomness** in feature selection and data sampling.
- **Reduces variance** compared to individual decision trees.
- **Performs well on high-dimensional data**.
- **Works for both classification and regression tasks**.

Random Forest Algorithm

Random Forests is the most well-known implementation of a **Forest of Randomized Trees**. It works as follows:

1. **Bootstrap Sampling**: Each tree is trained on a random subset of the dataset (sampling with replacement).
2. **Feature Randomization**: At each node, a random subset of features is considered for splitting.
3. **Tree Growth**: Each tree is grown to its maximum depth without pruning.
4. **Aggregation**:
    - → **Classification**: Uses majority voting across trees.
    - → **Regression**: Averages predictions from all trees.

Additional Variants

- → **Extremely Randomized Trees (Extra Trees)**: Unlike Random Forests, Extra Trees select split points randomly instead of finding the optimal split.
- → **Oblique Random Forests**: Use linear combinations of features to split nodes instead of single feature splits.

Benefits of Random Forests

- Robust to overfitting due to averaging across trees.
- Handles missing values effectively by using feature importance.
- Works well with categorical and numerical data.
- Computationally efficient due to parallel tree training.
- Provides feature importance ranking, aiding interpretability.
- Handles imbalanced datasets by weighting class occurrences in decision-making.

Hyperparameters of Random Forests

- **Number of Trees (n_estimators)**: More trees improve stability but increase computational cost.
- **Max Features**: Number of features considered for each split; common choices are `sqrt` (default for classification) and `log2`.
- **Max Depth**: Limits the depth of trees to prevent overfitting.
- **Min Samples Split**: Minimum number of samples required to split a node; default is 2.
- **Min Samples Leaf**: Minimum number of samples required to form a leaf node; higher values reduce overfitting.
- **Criterion**: Determines how to measure the quality of a split (e.g., **Gini impurity** or **Entropy** for classification, **Mean Squared Error** for regression).

Randomized Trees vs. Standard Decision Trees

Feature	Decision Trees	Random Forests
Overfitting	High	Low
Stability	Low	High
Computational Cost	Low	Higher
Performance on Large Datasets	Moderate	High
Handles Missing Data	No	Yes
Provides Feature Importance	No	Yes

Applications of Random Forests

- **Finance**: Credit scoring, risk analysis, stock price prediction.
- **Healthcare**: Disease prediction, medical imaging analysis, drug discovery.
- **E-commerce**: Customer segmentation, recommendation systems, sales forecasting.
- **Cybersecurity**: Intrusion detection, fraud detection, malware classification.
- **Natural Language Processing**: Sentiment analysis, document classification.
- **Remote Sensing**: Land cover classification, weather prediction.

Limitations of Random Forests

- ✘ **Computationally expensive for large datasets** due to multiple trees.
- ✘ **Less interpretable than single decision trees**.
- ✘ **Memory-intensive**, especially with large numbers of trees.
- ✘ **Not optimal for real-time predictions** since multiple trees must be queried.

Summary

- **Forest of Randomized Trees** improves model stability and reduces overfitting.
- **Random Forests** use bootstrapping and feature selection to enhance accuracy.
- **Extra Trees and Oblique Random Forests** introduce further randomness for diversity.
- **Well-suited for classification, regression, and high-dimensional problems**.
- **Hyperparameter tuning** is crucial for balancing accuracy and computational efficiency.

Boosting

Overview

Boosting is an ensemble learning technique that **combines multiple weak learners** to create a strong learner. Unlike bagging, where models are trained independently, boosting **sequentially trains models**, giving more importance to misclassified examples at each step.

Key Characteristics

- **Sequential Learning**: Each new model corrects the errors of the previous one.
- **Weighted Training**: Misclassified instances receive higher weights.
- **Reduces Bias**: Effective in converting weak models into strong models.
- **Prone to Overfitting**: Can be sensitive to noisy data.

Boosting Algorithm Steps

1. **Initialize weights**: Assign equal weights to all training samples.
2. **Train a weak classifier**: Typically a decision tree with depth = 1 (stump).
3. **Compute error**: Measure the performance of the weak classifier.
4. **Update sample weights**: Increase weights for misclassified samples.
5. **Train another weak classifier** using the updated weights.
6. **Repeat for a fixed number of iterations** or until the error is minimized.
7. **Aggregate the classifiers' outputs** to make a final prediction.

Popular Boosting Algorithms

1. AdaBoost (Adaptive Boosting)

- → **Concept**: Each weak learner is assigned a weight based on its accuracy.
- → **Training Process**:
  - Train weak classifiers sequentially.
  - Misclassified examples receive higher weights.
  - The final prediction is a weighted majority vote.
- → **Strengths**:
  - Simple and effective.
  - Reduces bias and variance.
- → **Weaknesses**:

- Sensitive to noisy data.
- Performance degrades with too many weak learners.

2. Gradient Boosting (GBM - Gradient Boosting Machines)

- **Concept**: Instead of adjusting sample weights, GBM **fits new models to the residual errors** of previous models.
- **Training Process**:
  - A weak model is trained to predict residuals (errors).
  - The next model is trained to correct these residuals.
  - This process continues until the error is minimized.
- **Strengths**:
  - More flexible than AdaBoost.
  - Can use different loss functions (e.g., squared error for regression, log loss for classification).
- **Weaknesses**:
  - Computationally expensive.
  - Requires careful tuning of hyperparameters.

3. XGBoost (Extreme Gradient Boosting)

- **Concept**: An optimized version of gradient boosting that is **faster and more efficient**.
- **Key Features**:
  - Uses **regularization** to prevent overfitting.
  - **Handles missing values automatically**.
  - Supports **parallel computation** for faster training.
- **Strengths**:
  - State-of-the-art performance on structured data.
  - Efficient and scalable.
- **Weaknesses**:
  - Requires careful hyperparameter tuning.
  - More complex to implement.

4. LightGBM (Light Gradient Boosting Machine)

- **Concept**: A gradient boosting method optimized for **speed and memory efficiency**.
- **Key Features**:
  - Uses **leaf-wise growth** instead of level-wise growth.

    - Efficient for large datasets.
- → **Strengths**:
    - Faster training time compared to XGBoost.
    - Low memory usage.
- → **Weaknesses**:
    - Can overfit on small datasets.
    - Requires specific preprocessing for categorical data.

5. CatBoost (Categorical Boosting)

- → **Concept**: A gradient boosting method optimized for handling **categorical data**.
- → **Key Features**:
    - Does not require explicit one-hot encoding.
    - Works well with imbalanced datasets.
- → **Strengths**:
    - High accuracy with categorical data.
    - Robust to missing values.
- → **Weaknesses**:
    - Slower training compared to LightGBM.
    - Requires careful tuning for optimal performance.

Comparison of Boosting Methods

Method	Speed	Overfitting Risk	Handles Missing Data	Requires Hyperparameter Tuning
AdaBoost	Medium	High	No	Low
Gradient Boosting	Slow	Medium	No	High
XGBoost	Fast	Low	Yes	High
LightGBM	Very Fast	Medium	Yes	High
CatBoost	Medium	Low	Yes	High

Advantages of Boosting

✔ Boosts weak models into strong learners.
✔ Handles both classification and regression tasks.
✔ Works well with imbalanced datasets.
✔ Improves accuracy on structured data.

Disadvantages of Boosting

✘ Computationally expensive for large datasets.
✘ Sensitive to noisy data (especially AdaBoost).
✘ Requires careful hyperparameter tuning.
✘ Can overfit if too many weak learners are used.

Applications of Boosting

- **Finance**: Credit scoring, fraud detection.
- **Healthcare**: Disease prediction, medical diagnosis.
- **Marketing**: Customer segmentation, churn prediction.
- **Cybersecurity**: Malware detection, anomaly detection.
- **NLP (Natural Language Processing)**: Sentiment analysis, spam filtering.

Summary

- **Boosting sequentially trains weak learners** to correct previous errors.
- **AdaBoost** assigns weights to misclassified examples.
- **Gradient Boosting** fits new models to residual errors.
- **XGBoost, LightGBM, and CatBoost** offer performance optimizations for large datasets.
- **Boosting improves accuracy but requires careful tuning** to avoid overfitting.

Comparison of Classification Methods

Method	Type	Pros	Cons	Best Use Cases
Naive Bayes	Probabilistic	Fast, works well with small datasets, interpretable	Assumes independence, struggles with correlated features	Text classification, spam detection
Perceptron	Linear Classifier	Simple, efficient	Only works with linearly separable data	Basic binary classification
SVM	Linear & Non-Linear	Effective in high dimensions, kernel trick for non-linearity	Computationally expensive for large datasets	Image recognition, bioinformatics
Neural Networks	Deep Learning	Powerful, learns complex patterns	Computationally expensive, requires large data	Speech recognition, NLP, computer vision
KNN	Instance-Based	Simple, no training phase	Slow for large datasets, sensitive to irrelevant features	Recommendation systems, anomaly detection
Ensemble Methods	Meta-Classifier	Improves accuracy, reduces overfitting	High computational cost, less interpretable	Finance, healthcare, cybersecurity
Random Forests	Ensemble	Reduces overfitting, interpretable	High computational cost for large datasets	Feature importance analysis, medical diagnostics
Boosting (XGBoost, AdaBoost, etc.)	Ensemble	Highly accurate, reduces bias	Sensitive to noisy data, requires tuning	Fraud detection, competition-level ML models

Key Takeaways

- ✓ **No single classifier is best for all tasks**; selection depends on data characteristics and use case.
- ✓ **Linear models (Perceptron, SVM) work well for linearly separable data**, but require kernel tricks for complex patterns.
- ✓ **Naive Bayes is efficient for probabilistic tasks**, especially in text processing.
- ✓ **Neural Networks excel in high-dimensional, unstructured data**, such as images and speech.
- ✓ **Ensemble methods (Bagging, Boosting, Random Forests) improve accuracy and stability**, but require more computational resources.

Choosing the Right Classifier

If you need	**Consider**
A fast, simple model	Naive Bayes, Perceptron
A robust model for structured data	Random Forest, Boosting
A method that works well with small data	KNN, Naive Bayes
A highly scalable approach	SVM, Boosting (XGBoost, LightGBM)
Best accuracy possible	Neural Networks, Boosting

Preprocessing

Data Type Conversions

Overview

Many machine learning algorithms require numerical features. Thus, categorical and ordinal features must be transformed into numeric representations. Some tasks also require binarization or discretization of continuous variables.

Why Do We Need Type Conversion?

1. **Algorithms Require Numeric Features**: Many ML models (e.g., linear regression, SVM) cannot process categorical data.
2. **Preserving Order in Ordinal Features**: Ordinal categories (e.g., poor < good < excellent) must maintain their ranking when converted.
3. **Transforming Numerical Targets for Classification**: Discretization allows regression targets to be used in classification tasks.
4. **Boolean Representation for Association Rule Mining**: Certain algorithms require boolean attributes derived from numerical data.

Methods for Type Conversion

1. Binarization of Discrete Attributes

A categorical attribute with V unique values is transformed into V binary attributes.

✓ Example:

Color	Color-Red	Color-Blue	Color-Green	Color-Orange	Color-Yellow
Red	1	0	0	0	0
Blue	0	1	0	0	0
Green	0	0	1	0	0

2. Nominal to Numeric (One-Hot Encoding)

Converts categorical variables into multiple binary variables.

- → Implemented in **scikit-learn** as `OneHotEncoder`.

3. Ordinal to Numeric (Ordinal Encoding)

Converts ordered categorical values into consecutive integers.

- → Example: `awful, poor, ok, good, great` → `0,1,2,3,4`.
- → Implemented using `OrdinalEncoder` in **scikit-learn**.

4. Numeric to Binary with Thresholding

Values below a threshold become `0`, and values above become `1`.

- → Implemented using `Binarizer` in **scikit-learn**.

5. Discretization (Reduction of Distinct Values)

Reduces continuous data to discrete bins to highlight patterns.

- Common methods:
  - → **Equal-width binning**: Divides range into equal-sized intervals.
  - → **Equal-frequency binning**: Each bin contains the same number of samples.
  - → **K-means binning**: Uses K-means clustering to determine bin edges.
- Implemented using `KBinsDiscretizer` in **scikit-learn**.

Summary

- **Type conversion is necessary** for ML models that require numeric inputs.
- **One-hot encoding** and **ordinal encoding** handle categorical data.
- **Binarization and discretization** help transform continuous data into structured formats.
- **scikit-learn provides built-in tools** (`OneHotEncoder`, `OrdinalEncoder`, `Binarizer`, `KBinsDiscretizer`).

Sampling

Overview

Sampling is used in machine learning when processing the entire dataset is impractical due to **computational cost or time constraints**. A well-chosen sample **preserves the key characteristics** of the dataset while reducing processing requirements.

Why Use Sampling?

1. **Large datasets may be too expensive to process entirely**.
2. **Sampling speeds up model training and evaluation**.
3. **Representative samples provide similar insights as the full dataset**.

Types of Sampling

1. Simple Random Sampling

Each data point has an **equal probability** of being selected.

→ Useful when **no prior knowledge** of data distribution is available.

2. Sampling With Replacement

Data points can be **selected multiple times**. **+** Commonly used in **bootstrapping** to generate multiple datasets from one sample.

3. Sampling Without Replacement

Each data point is selected **only once**.

- Ensures unique samples but may **reduce diversity** in smaller datasets.

4. Stratified Sampling

Ensures **each subgroup (stratum) of the dataset** is represented.

✓ Used when dataset contains **imbalanced classes**.
✓ Essential for **classification tasks with rare categories**.

5. Adaptive Sampling

Dynamically adjusts sampling based on **data characteristics**.

✓ Used for **data streams and evolving datasets**.

Determining Sample Size

a. **Larger samples** increase precision but require more computation.
b. **Statistical methods** (e.g., power analysis) help **determine optimal sample size**.
c. **Trade-off** between sample size and computational efficiency.

Sampling in Scikit-Learn

→ `train_test_split()` – Splits dataset into train/test sets (random sampling).
→ `StratifiedKFold()` – Ensures balanced class representation.
→ `resample()` – Generates bootstrap samples.

Summary

- Sampling is necessary when processing large datasets.
- Stratified sampling is important for maintaining class balance.
- With or without replacement affects how diverse samples are.
- Scikit-learn provides built-in sampling utilities to facilitate data partitioning.

Feature Creation

Overview

Feature creation is the process *of generating new features that better represent the underlying patterns in data*. It enhances model performance by providing more relevant information.

Why Feature Creation Matters

1. Enhances predictive power by capturing hidden patterns.
2. Transforms raw data into meaningful insights.
3. Reduces reliance on complex models by encoding information efficiently.

Types of Feature Creation

1. Feature Extraction

Converts raw data into more useful features.

- ✓ Examples:
  - → From image pixels → extract edge detectors, color histograms.
  - → From text data → extract word frequencies, **TF-IDF scores**.

2. Feature Mapping (Transformation to a New Space)

Projects data into a different representation.

- ✓ Example: Fourier Transform for signal processing.

3. Derived Features

Create new features based on existing ones.

- ✓ Examples:
  - → Finance: `Volume / Weight → Density`
  - → Housing: `Price / Square Foot`

Examples of Feature Creation in Finance

- ✓ Moving Averages: Calculate the average closing price over a time window.
- ✓ Volatility Metrics: Standard deviation of price fluctuations.
- ✓ Relative Strength Index (RSI): Detects overbought/oversold conditions.
- ✓ Liquidity Ratios: Measure bid-ask spread and trading volume.

Examples of Feature Creation in Housing Data

- ✓ Total Area: Sum of all room sizes.
- ✓ Price per Square Foot: House price divided by total area.
- ✓ Distance to City Center: Measures accessibility.
- ✓ Renovation Status: Binary feature indicating recent renovations.

Automated Feature Engineering

- Featuretools: A Python library for automatically generating new features.
- Polynomial Features (Scikit-learn): Creates interaction terms between features.

Summary

- Feature creation is crucial for improving model accuracy.
- New features can be derived, mapped, or extracted from raw data.
- Industry-specific feature engineering enhances domain-specific performance.
- Libraries like `Featuretools` and `Scikit-learn` assist in automating feature generation.

Data Transformations

Overview

Data transformations adjust feature values to improve the performance of machine learning models. Certain transformations enhance interpretability, comparability, and algorithm performance.

Why Data Transformation?

1. Different feature scales can distort results.
2. Some models assume normally distributed data.
3. Outliers can disproportionately impact distance-based algorithms.
4. Ensures gradient-based algorithms (e.g., neural networks, logistic regression) converge properly.

Common Data Transformation Techniques

1. Feature Rescaling

Rescaling ensures that all features contribute equally to distance-based models.

- **Standardization (Z-score Normalization):**

$$x' = \frac{x - \mu}{\sigma}$$

  - Centers data around 0 with unit variance.
  - Used for Gaussian-distributed data.
- **Min-Max Scaling:**

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  - Maps values to a 0-1 range.
  - Preferred when feature ranges need to be bounded (e.g., neural networks).
- **Robust Scaling:**
  - Uses median and interquartile range.
  - Less sensitive to outliers.

2. Log Transformation

- Reduces skewness in distributions with extreme values.
- Converts an exponential relationship into a linear one.

3. Power Transformation

- **Box-Cox Transformation** (for strictly positive values):

$$x' = \frac{x^{\lambda} - 1}{\lambda}$$

- **Yeo-Johnson Transformation** (for all real values):
  - Handles both positive and negative values.

4. Normalization

- **L1 Normalization:** Scales feature vectors so they sum to 1.
- **L2 Normalization:** Scales feature vectors so their Euclidean norm equals 1.
- Used in cosine similarity-based models (e.g., text classification).

Effect on Distance-Based Algorithms

KNN, K-Means, SVMs rely on feature distances.

✓ Example:
   a. **Before scaling:** One feature (e.g., salary) dominates distances.
   b. **After scaling:** Features contribute equally to distance calculations.

Scikit-Learn Solutions for Feature Rescaling

→ `StandardScaler`: Standardization (Z-score normalization).
→ `MinMaxScaler`: Min-max scaling.
→ `RobustScaler`: Scaling based on median/IQR.
→ `PowerTransformer`: Box-Cox and Yeo-Johnson transformations.
→ `Normalizer`: L1 and L2 normalization.

Summary

- Data transformation ensures fair feature contribution and model convergence.
- Feature rescaling (standardization, min-max, robust scaling) prevents dominance by large-scale features.
- Log and power transformations reduce skewness.
- Scikit-learn provides various preprocessing utilities for efficient feature scaling.

Imbalanced Data in Classification

Overview

Imbalanced data occurs when one class is significantly underrepresented compared to others, leading to biased model performance. Standard accuracy metrics can be misleading in these cases.

Challenges of Imbalanced Data

1. **Majority class dominates predictions**, **leading to poor recall for minority class.**
2. Traditional accuracy *is misleading* (e.g., 95% accuracy might mean predicting only the majority class correctly).
3. Decision boundaries may be skewed, making it harder to detect the minority class.

Strategies to Handle Imbalanced Data

1. Cost-Sensitive Learning

Adjusts model training to penalize misclassification of the minority class.

Some models allow setting `class_weight` (e.g., `class_weight='balanced'` in Scikit-learn classifiers).

→ Equivalent to oversampling the minority class during training.

2. Data Resampling Techniques

a. **Undersampling:** *Removes samples* from the **majority class.**
   ✘ Risk: May lose important information.
b. **Oversampling:** *Replicates* **minority class** examples.
   ✘ Risk: Increases overfitting.
c. **SMOTE** (Synthetic Minority Oversampling Technique): **Creates synthetic minority class samples** by interpolating between existing instances.

3. Algorithmic Approaches

a. **Tree-based models** (Random Forest, XGBoost) handle imbalance better due to bootstrapping.
b. Anomaly detection methods can be repurposed to detect rare class occurrences.
c. **Threshold adjustment:** Modify decision thresholds instead of retraining the model.

Performance Metrics for Imbalanced Data

- **Precision:** $\frac{TP}{TP + FP}$ (focuses on correct positive predictions).

- **Recall:** $\frac{TP}{TP + FN}$ (measures ability to find minority class instances).
- **F1-score:** Harmonic mean of precision and recall.
- **AUC-ROC Curve:** *Measures the trade-off between true positive rate and false positive rate.*

💡

*The **AUC-ROC Curve** (Area Under the Curve - Receiver Operating Characteristic) is a metric used to evaluate the performance of a classifier, especially when dealing with **imbalanced datasets**.*

How does it work?

a. ***True Positive Rate (TPR)**: Measures the model's ability to correctly identify actual positives (**also known as recall or sensitivity**).*
b. ***False Positive Rate (FPR)**: Measures the proportion of false positives relative to the total number of actual negatives.*

*The **AUC (Area Under the Curve)** represents the area under the ROC curve and ranges between **0 and 1**:*

- ***AUC = 1** → Perfect model.*
- ***AUC = 0.5** → Random model (**equivalent to flipping a coin**).*
- ***AUC < 0.5** → Worse than random guessing (**probably due to incorrect training**).*

*The higher the **AUC**, the **better the model's ability to distinguish between classes**. This metric is particularly useful for **imbalanced datasets**, as it assesses the model's predictive quality across different decision thresholds.*

Scikit-Learn Solutions

→ `class_weight='balanced'` in classifiers like `LogisticRegression`, `RandomForestClassifier`.
→ `imblearn.over_sampling.SMOTE()` for oversampling.

→ `imblearn.under_sampling.RandomUnderSampler()` for undersampling.

Summary

- Imbalanced data skews predictions towards the majority class.
- Resampling (oversampling, undersampling, SMOTE) helps balance training data.
- Cost-sensitive learning and model-specific techniques mitigate imbalance.
- F1-score and AUC-ROC are preferred metrics over accuracy.
- Scikit-learn and Imbalanced-learn (`imblearn`) offer built-in solutions.

Feature Selection

Overview

Feature selection is the process of identifying the most relevant features to improve model performance, reduce overfitting, and enhance interpretability.

Why Feature Selection Matters

1. Speeds up training by reducing dimensionality.
2. Improves model accuracy by removing irrelevant/noisy features.
3. Reduces overfitting, leading to better generalization.
4. Enhances interpretability by focusing on meaningful attributes.

Types of Feature Selection

1. Filter Methods

Select features based on statistical relevance.

✓ Common techniques:
  - **Pearson's Correlation:** Measures linear dependence between features.
  - **ANOVA** (Analysis of Variance): Evaluates differences between feature groups.

- **Chi-Square Test:** Assesses correlation between categorical features.
- **Mutual Information:** Measures how much a feature contributes to predicting the target.

Fast and independent of the model but does not account for feature interactions.

2. Wrapper Methods

Iteratively train models on feature subsets and evaluate performance.

✓ Common approaches:
- **Recursive Feature Elimination (RFE):** Removes least important features in steps.
- **Forward Selection:** Starts with no features and adds the most useful ones.
- **Backward Elimination:** Starts with all features and removes the least useful ones.

More accurate but computationally expensive.

3. Embedded Methods

Feature selection is performed during model training.

✓ Common techniques:
- **Lasso Regression:** Uses L1 regularization to shrink coefficients of less relevant features to zero.
- **Decision Trees & Random Forests:** Assign feature importance scores based on splits.

Balances efficiency and accuracy but is model-dependent.

Feature Selection in Scikit-Learn

- `SelectKBest()`: Chooses the top k best features using statistical tests.
- `RFE()`: Implements Recursive Feature Elimination.
- `LassoCV()`: Uses cross-validation to select features via L1 regularization.
- `FeatureImportances_`: Extracts feature importance scores from tree-based models.

Summary

- Feature selection enhances model efficiency and accuracy.
- Filter, wrapper, and embedded methods offer different trade-offs between speed and accuracy.
- Scikit-learn provides built-in feature selection tools for practical implementation.
- Choosing the right method depends on dataset size and computational constraints.

Dimensionality Reduction

Overview

Dimensionality reduction aims to reduce the number of features while retaining as much relevant information as possible. This helps improve model performance, reduce overfitting, and enhance interpretability.

Why Dimensionality Reduction?

1. Avoids the curse of dimensionality, where high-dimensional data becomes sparse and less informative.
2. Speeds up computations by reducing feature space complexity.
3. Removes redundant and correlated features.
4. Improves visualization of data by projecting it into lower dimensions.

Common Dimensionality Reduction Techniques

1. Principal Component Analysis (PCA)

Finds a new orthogonal feature space that captures maximum variance.

✓ **Steps:**
   1. Compute the covariance matrix of the dataset.
   2. Perform eigenvalue decomposition to obtain principal components.
   3. Select the top-k components that capture most of the variance.

- **Computational Complexity:**

- $O(ND^2)$ if $D \leq N$ (number of features ≤ number of samples).
- $O(N^2D)$ if $D > N$ (dual PCA approach).

Used for data compression and noise reduction.

2. Multi-Dimensional Scaling (MDS)

Projects high-dimensional data into a lower-dimensional space while preserving pairwise distances.

- **Two types:**
  1. **Classical MDS:** Uses eigenvalue decomposition.
  2. **Non-metric MDS:** Preserves the rank order of distances rather than exact values.

Used for data visualization and similarity analysis.

3. t-SNE (t-Distributed Stochastic Neighbor Embedding)

→ Projects data into 2D or 3D for visualization.
→ Retains local structure by maintaining distances between similar points.
→ Used primarily for clustering and exploratory data analysis.

4. Autoencoders (Neural Networks for Dimensionality Reduction)

Unsupervised deep learning technique that encodes data into a lower-dimensional representation.

✓ Encoder compresses data → bottleneck layer → decoder reconstructs original data.

Used for non-linear feature extraction.

Scikit-Learn Solutions for Dimensionality Reduction

- `PCA()`: Implements Principal Component Analysis.
- `MDS()`: Multi-Dimensional Scaling.
- `TSNE()`: t-SNE for visualization.
- `TruncatedSVD()`: Efficient PCA alternative for sparse data.

Summary

- **Dimensionality reduction** removes redundant features and improves efficiency.
- PCA, MDS, t-SNE, and autoencoders are common techniques.
- Choosing the right method depends on interpretability vs. computational cost.
- Scikit-learn provides various tools for dimensionality reduction and visualization.

The Scikit-Learn Solution for Feature Selection

Overview

Scikit-learn provides several tools for **automatic feature selection**, allowing models to focus on the most relevant variables and improve efficiency.

Methods for Feature Selection in Scikit-Learn

1. Filter Methods

Select features based on **statistical scores**.

✓ Implemented using:
- `SelectKBest()`: Chooses the top **k** features using statistical tests.
- `SelectPercentile()`: Selects a percentage of the best features.
- `mutual_info_classif()`, `f_classif()`, `chi2()`: Compute feature importance scores.

2. Wrapper Methods

Iteratively add or remove features based on model performance.

✓ Implemented using:
- `RFE() (Recursive Feature Elimination)`: Recursively removes least important features.
- `RFECV()`: Similar to RFE but uses **cross-validation** to optimize feature selection.

3. Embedded Methods

Perform feature selection **during model training**.

✓ Implemented using:

- `Lasso (L1 regularization)`: Shrinks less important feature coefficients to zero.
- `RandomForestClassifier.feature_importances_`: Uses decision tree splits to rank features.
- `SelectFromModel()`: General method that selects important features from any trained model.

When to Use Each Method

Method	Best Use Case
Filter Methods	Large datasets, quick preprocessing
Wrapper Methods	Small datasets, improving specific model performance
Embedded Methods	Feature selection during model training

✓ **Example:** Using Feature Selection in Scikit-Learn

```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y = True)
selector = SelectKBest(score_func = f_classif, k = 2)
X_new = selector.fit_transform(X, y)
print(selector.scores_)
```

Summary

- Scikit-learn provides multiple feature selection methods (filter, wrapper, embedded).
- Filter methods are fast but do not account for feature interactions.
- Wrapper methods improve feature selection but are computationally expensive.
- Embedded methods integrate selection directly within models.
- Choosing the right method depends on dataset size and computational constraints.

Outlier-Detection

Outlier Detection - Overview

Definition and Importance

Outlier detection identifies **data points that significantly deviate** from the majority of the dataset. These anomalies can indicate:

- **Noise or errors** in the data.
- **Rare but significant events** (e.g., fraud, equipment failure).
- **Variability in the process being analyzed**.

Outlier detection is widely used in:

- **Fraud detection** (unusual transaction patterns).
- **Predictive maintenance** (detecting machine failures).
- **Healthcare** (identifying rare disease cases).
- **Finance** (detecting trading irregularities).
- **Network security** (identifying potential cyberattacks).

Key Techniques for Outlier Detection

1. **Statistical Methods**:

    - Assume a predefined **probability distribution** for the data.
    - Common techniques:
        - → **Z-Score**: Measures the distance from the mean in standard deviations.
        - → **Interquartile Range (IQR)**: Uses quartiles to detect outliers.
2. **Distance-Based Methods**:

    - Measure the **distance** of each data point from its neighbors.
    - Common techniques:
        - → **k-Nearest Neighbors (k-NN)**: Anomalies have higher distances from neighbors.
        - → **DBSCAN**: Identifies outliers as points that do not belong to any dense cluster.

3. **Model-Based Methods**:

    - Train machine learning models to differentiate normal and abnormal data.
    - Common techniques:
        - → **Isolation Forest**: Randomly partitions data to isolate outliers.
        - → **Autoencoders**: Neural networks that reconstruct normal data and detect anomalies.

Challenges in Outlier Detection

- **High Dimensionality**: Outliers become harder to detect as the number of features increases (curse of dimensionality).
- **Imbalanced Data**: Outliers are rare, making standard classification methods ineffective.
- **Noise in Data**: Differentiating true anomalies from random noise is difficult.
- **Dynamic Datasets**: Continuous data streams require adaptive methods.

Evaluation Metrics for Outlier Detection

To assess the effectiveness of an outlier detection method, common metrics include:

- → **Precision**: Fraction of detected outliers that are true outliers.
- → **Recall**: Fraction of true outliers correctly detected.
- → **F1-Score**: Harmonic mean of precision and recall.
- → **Area Under the Curve (AUC)**: Evaluates model performance across different thresholds.
- → **Execution Time**: Important for real-time applications like fraud detection.

Definition and Importance

Outlier detection identifies **data points that significantly deviate** from the majority of the dataset. These anomalies can indicate:

- **Noise or errors** in the data.
- **Rare but significant events** (e.g., fraud, equipment failure).
- **Variability in the process being analyzed**.

Outlier detection is widely used in:

- **Fraud detection** (unusual transaction patterns).
- **Predictive maintenance** (detecting machine failures).
- **Healthcare** (identifying rare disease cases).
- **Finance** (detecting trading irregularities).
- **Network security** (identifying potential cyberattacks).

Key Techniques for Outlier Detection

1. **Statistical Methods**:
    - Assume a predefined **probability distribution** for the data.
    - Common techniques:
        - → **Z-Score**: Measures the distance from the mean in standard deviations.
        - → **Interquartile Range (IQR)**: Uses quartiles to detect outliers.
2. **Distance-Based Methods**:
    - Measure the **distance** of each data point from its neighbors.
    - Common techniques:
        - → **k-Nearest Neighbors (k-NN)**: Anomalies have higher distances from neighbors.
        - → **DBSCAN**: Identifies outliers as points that do not belong to any dense cluster.
3. **Model-Based Methods**:
    - Train machine learning models to differentiate normal and abnormal data.
    - Common techniques:

- **Isolation Forest**: Randomly partitions data to isolate outliers.
- **Autoencoders**: Neural networks that reconstruct normal data and detect anomalies.

Challenges in Outlier Detection

- **High Dimensionality**: Outliers become harder to detect as the number of features increases (curse of dimensionality).
- **Imbalanced Data**: Outliers are rare, making standard classification methods ineffective.
- **Noise in Data**: Differentiating true anomalies from random noise is difficult.
- **Dynamic Datasets**: Continuous data streams require adaptive methods.

Evaluation Metrics for Outlier Detection

To assess the effectiveness of an outlier detection method, common metrics include:

- **Precision**: Fraction of detected outliers that are true outliers.
- **Recall**: Fraction of true outliers correctly detected.
- **F1-Score**: Harmonic mean of precision and recall.
- **Area Under the Curve (AUC)**: Evaluates model performance across different thresholds.
- **Execution Time**: Important for real-time applications like fraud detection.

Problem Description

Anomaly Detection

Anomaly detection aims to identify objects that differ significantly from most other objects in a dataset. The challenge is to determine how to measure dissimilarity and explore the dataset to uncover such anomalies.

Key Considerations

- Anomalies do not necessarily imply a small number of instances.
- Data collection errors can also introduce anomalies.
- Anomaly vs. Outlier: The terms are often used interchangeably.

Applications of Anomaly Detection

- Fraud Detection: Identifying unusual transactions (e.g., credit card fraud).
- Network Intrusion Detection: Monitoring network traffic for attacks.
- Ecosystem Disturbances: Predicting natural disasters like hurricanes.
- Medical Diagnosis: Detecting rare diseases based on symptoms.
- Public Health: Identifying anomalies in disease outbreaks.

Causes of Anomalies

1. **Data from Different Classes:**

    - Some anomalies represent a completely different category.
    - Example: Fraudulent transactions vs. normal transactions.
    - Hawkins' Definition of an Outlier:
        - → *An outlier is an observation that differs so much from others that it suggests a different generating mechanism.*
2. **Natural Variation:**

    - Extreme values in normally distributed data may be interesting but rare.
    - Example: A person with an extremely high IQ.

3. **Data Measurement and Collection Errors:**

    - Mistakes in data entry, faulty sensors, or missing values.
    - Such anomalies are usually addressed via data cleaning.

Approaches to Anomaly Detection

1. **Model-Based Techniques:**
   Build a statistical model (e.g., probability distributions, regression models). **+** Outliers are detected as poorly fitting data points.
    ✓ Example: A regression model where an outlier deviates far from predicted values.
2. **Proximity-Based Techniques:**
   Anomalous points are those that are distant from most other points. **+** Scatter plots help visualize anomalies in 2D/3D spaces.
3. **Density-Based Techniques:**
   Define anomalies as points in low-density regions. **+** Density is measured through kernel density estimation or neighbor-based methods.

The Role of Class Labels

1. **Supervised Anomaly Detection:**

    - Requires labeled datasets with both normal and anomalous instances.
    - Problem: Class imbalance (anomalies are rare, making classification difficult).
2. **Unsupervised Anomaly Detection:**

    - No labeled data available.
    - The algorithm assigns an anomaly score to each data point.
    - Assumes anomalies are sufficiently different from normal data.
3. **Semi-Supervised Anomaly Detection:**

    - Training set contains only normal data.
    - Anomalies are detected when new instances deviate significantly.
    - Also known as one-class classification.

Key Issues in Anomaly Detection

- **Number of Attributes:** Anomalies may be in single attributes or in attribute combinations.
- **Global vs. Local Perspective:** Anomalies might be global or relative to their neighborhood.
- **Degree of Anomaly:** Instead of a binary decision, some methods assign anomaly scores.
- **Masking and Swamping:**
    - **Masking:** Multiple anomalies hide each other.
    - S**wamping:** Anomalies distort the model, making normal objects appear as outliers.
- **Efficiency:** Some methods have high complexity (e.g., proximity-based methods have $O(N^2)$ complexity).

Summary

- Anomalies can arise from real variations, errors, or different processes.
- Different detection methods exist, including model-based, proximity-based, and density-based approaches.
- The choice of method depends on data availability, labeled information, and computational constraints.

Statistical Approaches to Outlier Detection

Overview

Statistical methods detect outliers by assuming that **data follows a specific probability distribution**. Anomalies are identified as **data points that deviate significantly** from this distribution.

Probabilistic Definition of an Outlier

- An **outlier** is a data point with **low probability** under a given probability model.
- The probability distribution model is estimated from the data and tested against observed values.
- Common **statistical tests** are used to identify discordant observations.

Issues in Probabilistic Definitions

- **Selecting the correct distribution**: The model can be **Gaussian, Poisson, Binomial**, etc. A wrong model invalidates results.
- **Univariate vs. Multivariate data**: More complex techniques are required for multiple attributes.
- **Mixtures of Distributions**: Some datasets contain multiple underlying distributions, requiring techniques like **Gaussian Mixture Models (GMM)**.

Detecting Outliers in a Normal Distribution

For a **univariate normal distribution** $N(\mu, \sigma^2)$, a point x is considered an outlier if:

$$|x - \mu| \geq c\sigma$$

where c is a threshold determined based on a desired **false positive rate** α.

Common Statistical Methods

1. **Z-Score (Standard Score Method)**
   Measures how many **standard deviations** a point is from the mean.
   - A data point x is an outlier if: $Z = \frac{x-\mu}{\sigma} \geq c$
   - Common thresholds:
     - c = 2.5 (mild outliers)
     - c = 3 (strong outliers)
2. **Interquartile Range (IQR)**
   Uses quartiles to identify extreme values.
   - The **IQR** is defined as: $IQR = Q_3 - Q_1$
   - Outliers are points outside:
     $[Q_1 - 1.5 \times IQR, Q_3 + 1.5 \times IQR]$
   - More robust than Z-score for **non-Gaussian data**.
3. **Grubbs' Test**
   A formal test for outliers in **normally distributed** data.

Tests whether the **largest absolute deviation** is significantly different from the mean.

- → Uses a t-distribution to determine statistical significance.

4. **Chi-Square Test for Multivariate Data**

   - Measures deviation of multiple attributes from their expected values.
   - A point is an outlier if: $D^2 = (x - \mu)^T \Sigma^{-1} (x - \mu) > c$
   - Where Σ is the covariance matrix and cc is a threshold from the Chi-square distribution.

Strengths and Weaknesses

✔ Strong theoretical foundation with well-established methods.
✔ Effective for small, normally distributed datasets.
✔ Z-Score and IQR are computationally efficient.

✘ Assumes a specific data distribution, which may not hold for real-world datasets.
✘ Less effective for high-dimensional or multimodal data.
✘ Fails when data has multiple overlapping distributions.

Summary

- → Statistical approaches assume data follows a predefined probability model.
- → Z-Score and IQR are common techniques for detecting outliers.
- → Grubbs' Test and Chi-Square tests are useful for single and multiple attributes.
- → More advanced techniques (e.g., GMMs) handle multimodal distributions better.

Proximity-Based Outlier Detection

Overview

Proximity-based methods define outliers as **data points that are distant from most other points** in the dataset. These methods assume that normal points are located **close to their neighbors**, while anomalies lie far away.

Key Concepts

1. **Proximity Measure**: Defines how "far" or "close" data points are.
2. **Neighborhood Definition**: Determines how many points should be considered for comparison.
3. **Outlier Score**: Measures how much a point deviates from its neighborhood.

Methods for Proximity-Based Outlier Detection

1. Distance to k-Nearest Neighbors (k-NN Outlier Score)

The **outlier score** of a data point is defined as the distance to its **k-th nearest neighbor**.

→ A **larger distance** implies a **higher likelihood** of being an outlier.

The choice of **k** affects results:

- **Small k**: Outliers in sparse regions have low scores (false negatives).
- **Large k**: Dense clusters may distort results (false positives).

2. Distance-Based Outlier Definition (DB-Outlier)

Proposed by **Knorr and Ng (1998)**. A point is an **outlier** if **fewer than k** points exist within a given radius RR.

Formal definition: $|\{y \in D : dist(x, y) \leq R\}| < k$ where:

- x is the data point.
- D is the dataset.
- $dist(x, y)$ is the distance between points.

- R is the predefined distance threshold.
- k is the minimum number of required neighbors.

3. Proximity-Based Solutions for Finding Outliers

a. **Brute Force Approach**:
   - → Compute pairwise distances for all **N** points.
   - → Complexity: $O(N^2)$, making it inefficient for large datasets.

b. **Optimized Algorithms**:
   - → Use indexing structures (e.g., KD-Trees, Ball Trees) to reduce computational cost.
   - → Reduce complexity to O(NlogN) in low dimensions.

Strengths and Weaknesses

✔ Simple and intuitive concept.
✔ Effective for detecting global outliers.
✔ Works well in low-dimensional spaces.

✘ Sensitive to distance metric selection (Euclidean, Manhattan, etc.).
✘ Computationally expensive for large datasets without optimizations.
✘ Fails with varying density distributions (outliers in dense regions are ignored).

Summary

- **Proximity-based methods identify anomalies based on distance to neighbors**.
- **k-NN Outlier Score** ranks points based on distances to neighbors.
- **DB-Outlier** detects outliers with fewer than **k neighbors within a radius R**.
- **Efficient implementations** use tree-based indexing to speed up computations.

Density-Based Outlier Detection

Overview

Density-based methods identify outliers as data points in low-density regions. These methods assume that normal data points belong to dense clusters, while anomalies exist in sparse areas.

Key Concepts

1. Density Estimation: Measures how densely populated a region is.
2. Outlier Score: Inverse of the density around a point.
3. Local vs. Global Perspective: Some methods compare local densities to detect outliers.

Methods for Density-Based Outlier Detection

1. Inverse Distance-Based Density

Defines density as the inverse of the average distance to the kk-nearest neighbors:

$$density(x, k) = \left(\frac{\Sigma_{y \in N_k(x)} dist(x,y)}{k}\right)^{-1}$$

- N_k= set of k-nearest neighbors.
- Lower density implies a higher likelihood of being an outlier.

2. Count-Based Density Estimation

Defines density as the number of points within a given radius R:

$$density(x, R) = |\{y \in D : dist(x, y) \leq R\}|$$

Challenges:

→ Choice of R is critical.
→ If R is too small, density may be underestimated.
→ If R is too large, density may be overestimated.

3. Local Outlier Factor (LOF)

Compares the density of a point with the densities of its neighbors.

The LOF score is calculated as:

$$LOF_k(x) = \frac{\sum_{y \in N_k(x)} \frac{density(y)}{density(x)}}{|N_k(x)|}$$

→ LOF > 1: x is in a sparser region than its neighbors (potential outlier).

→ LOF ≈ 1: x has a similar density to its neighbors (likely normal).

→ LOF < 1: x is denser than its neighbors (unlikely outlier).

Isolation Forest

Isolation Forest (iForest) is a *density-based* anomaly detection method that isolates anomalies using random partitions.

Algorithm Steps

1. Randomly split data along a feature dimension.
2. Repeat splits until each point is isolated in its own partition.
3. Anomalies require fewer splits (shorter path length in trees).
4. Compute anomaly score based on path length:

$$s(x) = 2^{-\frac{E(h(x))}{c(n)}}$$

where E(h(x)) is the average path length and c(n)) is a normalizing factor.

Advantages of Isolation Forest

✔ Fast and scalable (linear complexity: O(nlogn).
✔ Works well with high-dimensional data.
✔ Unsupervised, requires no prior assumptions about data distribution.

Disadvantages of Isolation Forest

✘ Random splits may not always isolate anomalies optimally.
✘ Assumes anomalies are few and different from normal data.

Strengths and Weaknesses of Density-Based Methods

✔ Detects outliers in datasets with varying density.
✔ Works well for non-globally distributed outliers (local anomalies).
✔ More robust to noise than distance-based approaches.

✘ Sensitive to hyperparameter selection (e.g., kk, RR, number of trees in iForest).
✘ Computationally expensive for high-dimensional data.
✘ Difficult to interpret compared to simpler statistical approaches.

Summary

- Density-based methods define outliers as points in low-density regions.
- Inverse distance, count-based density, and LOF are commonly used approaches.
- Isolation Forest isolates anomalies using recursive random splits.
- Choosing the right parameters is critical for effective anomaly detection.

Clustering

Introduction to Clustering

Overview

Clustering is an **unsupervised learning** technique that groups data points into **clusters** based on their similarity. Unlike classification, clustering does **not use labeled data**, meaning the algorithm must discover inherent structures in the dataset.

The Clustering Problem

Given:

- A dataset with N objects x_i, each described by D features x_{id}.
- The task is to find a **natural partitioning** into KKK clusters, possibly identifying **outliers** (noise objects).
- The output is a **clustering function**, which assigns each data point to a cluster.

Desirable Properties of Clusters

Objects within the same cluster should be **similar**. The clustering function should **maximize intra-cluster similarity** and **minimize inter-cluster similarity**.

Formal Definition

A clustering function **clust(·)** is defined as:

$$clust: X \rightarrow \{1, 2, \ldots, K\}$$

Such that:

- If x_1 and x_2 are **similar**, they belong to the same cluster:

$$clust(x_1) = clust(x_2)$$

- If x_1 and x_2 are **not similar**, they belong to different clusters:

$$clust(x_1) \neq clust(x_2)$$

Measuring Clustering Quality

To determine the best clustering scheme, we need **evaluation metrics**. Some ideas include:

1. **Sum of distances** between a point and others in its cluster.
2. **Average distance** within a cluster.
3. **Sum of squared distances** (penalizes sparse clusters).
4. **Centroid-based similarity**:
    - Define a **representative point** for each cluster.
    - Compute **distances** between data points and the cluster centroid.

Centroid

The centroid of a cluster k is the **mean** of all points assigned to it:

$$centroid_k^d = \frac{1}{|\{x_i : clust(x_i) = k\}|} \sum_{x_i : clust(x_i) = k} x_{id}$$

This is also known as the **center of gravity** in physics.

Types of Clustering Methods

Clustering methods can be categorized as follows:

1. Partitioning Methods

The dataset is divided into KKK non-overlapping clusters.

✓ Examples:
- → **K-Means** (MacQueen 1967)
- → **Expectation Maximization (EM)** (Lauritzen 1995)
- → **CLARANS** (Ng & Han 1994)

2. Hierarchical Clustering

Creates a **tree-like structure** of nested clusters. Can be **agglomerative** (bottom-up) or **divisive** (top-down).

- ✓ Examples:
  - → **Agglomerative clustering**
  - → **BIRCH** (Zhang et al. 1996)
  - → **CURE** (Guha et al. 1998)

3. Density-Based Clustering

Finds clusters based on **dense regions** separated by low-density areas.

- ✓ Examples:
  - → **DBSCAN** (Ester et al. 1996)
  - → **DENCLUE** (Hinnenburg & Keim 1998)

4. Model-Based Clustering

Assumes that data is generated by a mixture of **probability distributions**.

- ✓ Examples:
  - **Gaussian Mixture Models (GMM)**
  - **COBWEB** (Fisher 1987)
  - **AutoClass** (Cheeseman 1996)

K-Means Clustering

Overview

K-Means is one of the most widely used **partitioning-based clustering algorithms**. It aims to partition a dataset into **K clusters**, where each data point belongs to the cluster with the nearest **centroid**.

Algorithm Steps

1. **Choose K**: Define the number of clusters.
2. **Initialize Centroids**: Select K random points as initial cluster centroids.
3. **Assign Points to Clusters**:
   - Each data point is assigned to the closest centroid.

4. **Update Centroids**:
    - Recalculate each centroid as the **mean of all points** assigned to that cluster.
5. **Repeat Steps 3 and 4** until centroids do not change (convergence).

Mathematical Formulation

The objective is to **minimize the distortion** (also called **inertia**), defined as:

$$SSE = \sum_{J=1}^{K} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

Where:

- K = Number of clusters.
- x_i = Data point.
- μ_j = Centroid of cluster jj.
- C_j = Set of points in cluster jj.
- SSE = **Sum of Squared Errors** (distortion measure).

Initialization Methods

The **initial placement of centroids** affects the final clustering result. Common strategies include:

- **Random Initialization**: Select K random points (can lead to poor results).
- **K-Means++**: Improves initialization by selecting diverse starting points.
- **Forgy Method**: Assigns K random points as initial centroids.
- **Random Partitioning**: Randomly assigns data points to clusters and computes initial centroids.

Convergence and Termination

- K-Means **always converges**, but it may reach a **local minimum**.
- Running K-Means multiple times with different initializations can help find a **better clustering**.
- The algorithm stops when:

1. Centroids do not change.
2. A maximum number of iterations is reached.

Choosing the Number of Clusters (K)

The optimal number of clusters is **not always known** and must be determined using:

- **Elbow Method**: Plot SSE vs. K and look for an "elbow" where SSE stops decreasing significantly.
- **Silhouette Score**: Measures how similar each point is to its own cluster compared to other clusters.
- **Gap Statistic**: Compares clustering results to a random dataset.

Advantages of K-Means

✔ **Simple and fast** for large datasets.
✔ **Interpretable and scalable**.
✔ **Works well for spherical clusters**.
✔ **Can be efficiently implemented** (e.g., MiniBatch K-Means for large datasets).

Disadvantages of K-Means

✘ **Requires specifying K** in advance.
✘ **Sensitive to outliers** and **noisy data**.
✘ **Does not handle non-convex clusters well**.
✘ **Assumes equal-sized, spherical clusters**.

Applications of K-Means

- **Image Segmentation**: Color quantization, object detection.
- **Customer Segmentation**: Marketing and recommendation systems.
- **Anomaly Detection**: Identifying unusual data points.
- **Biology**: Gene expression analysis.
- **Compression**: Vector quantization in data compression.

Evaluation of a Clustering Scheme

Overview

Evaluating a clustering scheme is crucial to assess cluster quality and compare different clustering methods. Since clustering is an unsupervised learning task, evaluation is based on intrinsic measures (cohesion, separation) or external ground truth comparisons.

Key Evaluation Metrics

1. Cohesion and Separation

- **Cohesion:** Measures how closely related points are within the same cluster.
- **Separation:** Measures how distinct clusters are from one another.
- **Total Sum of Squares (TSS):** Defined as: **TSS=SSE+SSB**
  - → SSE (Sum of Squared Errors): Intra-cluster variance (lower is better).
  - → SSB (Sum of Squares Between Clusters): Inter-cluster separation (higher is better).
  - → The goal is to minimize SSE while maximizing SSB.

2. Silhouette Score

Measures how similar an object is to its own cluster compared to other clusters.

Defined as: $s_i = \frac{b_i - a_i}{max(a_i, b_i)}$

- → a_i = Average intra-cluster distance (cohesion).
- → b_i = Minimum average inter-cluster distance (separation).
- → s_i ranges from -1 to 1 (higher values indicate better clustering).

3. Elbow Method

Used to find the optimal number of clusters (K). Plot SSE vs. K and look for an "elbow" where SSE stops decreasing significantly.

4. Supervised Measures (If Labels Are Available)

- **Confusion Matrix:** Compares clustering results with ground truth labels.
- **Precision, Recall, and F1-score:** Used when clusters should match predefined categories.
- **Adjusted Rand Index (ARI):** Measures similarity between predicted clusters and ground truth labels.
- **Jaccard Coefficient:** Measures how much two clustering schemes agree.

Choosing the Best Clustering Scheme

- → **Visual Inspection:** Plotting clusters in 2D or 3D (if possible).
- → **Comparing Silhouette Scores:** Higher scores indicate better separation.
- → **Trade-off Between Cohesion and Separation:** The best scheme balances compact clusters and clear separation.
- → Use domain knowledge to determine the meaningfulness of clusters.

Summary

- Clustering evaluation is challenging due to the lack of labels.
- Cohesion (SSE) and Separation (SSB) are key measures.
- Silhouette Score provides an intuitive measure of clustering quality.
- Supervised measures can be used if ground truth labels are available.
- The Elbow Method helps find the best K in partitioning methods.

Hierarchical Clustering

Overview

Hierarchical clustering builds a tree-like structure (dendrogram) to group data points into nested clusters. Unlike partitioning methods (e.g., K-Means), hierarchical clustering does not require specifying the number of clusters (K) in advance.

Types of Hierarchical Clustering

1. Agglomerative (Bottom-Up)
    - Starts with each point as its own cluster.
    - Iteratively merges the closest clusters until only one remains.
    - Most commonly used method.
2. Divisive (Top-Down)
    - Starts with all points in one cluster.
    - Recursively splits the clusters until each point is a separate cluster.

Distance (Linkage) Metrics

To determine how clusters are merged or split, different linkage criteria are used:

- **Single Linkage:** Distance between the closest points of two clusters.
- Complete Linkage: Distance between the farthest points of two clusters.
- Average Linkage: Mean distance between all pairs of points from two clusters.
- Ward's Method: Minimizes variance within clusters.

Algorithm (Agglomerative Clustering)

1. Compute the distance matrix between all points.
2. Initialize each point as a separate cluster.
3. Find the two closest clusters based on the linkage criterion.
4. Merge them into one cluster.
5. Repeat steps 3-4 until only one cluster remains.
6. Use the dendrogram to determine the final number of clusters.

Cutting the Dendrogram

- The dendrogram represents the hierarchy of clusters.
- By cutting the dendrogram at a certain level, a desired number of clusters can be obtained.
- The choice of the cut level is application-dependent and can be guided by measures like Silhouette Score.

Advantages of Hierarchical Clustering

✔ No need to specify K beforehand.
✔ Dendrogram provides interpretability of cluster relationships.
✔ Works well with non-convex clusters.
✔ Can handle different cluster sizes and densities.

Disadvantages of Hierarchical Clustering

✘ Computationally expensive (time complexity $O(N^3)$ for naive implementations).
✘ Sensitive to noise and outliers.
✘ Difficult to scale for very large datasets.
✘ Merging/splitting decisions are irreversible.

Applications of Hierarchical Clustering

- Bioinformatics: Gene expression analysis.
- Document Clustering: Grouping similar texts.
- Marketing: Customer segmentation.
- Image Processing: Object recognition.
- Social Networks: Community detection.

Summary

- Hierarchical clustering creates a tree-like structure of clusters.
- Agglomerative (bottom-up) and divisive (top-down) approaches exist.
- Different linkage criteria affect cluster formation.
- Dendrograms allow flexible cluster selection.
- Useful for structured data analysis but computationally expensive.

Density-Based Clustering

Overview

Density-based clustering identifies clusters as dense regions of points separated by low-density regions. Unlike partitioning and hierarchical methods, density-based algorithms can detect arbitrary-shaped clusters and are robust to noise.

Key Characteristics

- Clusters are dense regions of points surrounded by sparser areas.
- Does not require specifying the number of clusters (K) in advance.
- Can identify noise (outliers).
- Works well for clusters of varying shapes and sizes.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

One of the most popular density-based clustering methods.

Algorithm Steps

1. **Define parameters:**
    - ε(ε) = Neighborhood radius.
    - minPts = Minimum number of points required to form a dense region.
2. **Classify points:**
    - Core Points: Have at least minPts neighbors within ε\varepsilon.
    - Border Points: Have fewer than minPts neighbors but are reachable from a core point.
    - Noise Points: Are neither core nor border points.
3. **Expand clusters:**
    - Select an unvisited core point, form a new cluster.
    - Recursively add density-reachable points to the cluster.
    - Repeat until all points are classified.

Advantages of DBSCAN

✔ Detects arbitrary-shaped clusters.
✔ Does not require specifying K beforehand.
✔ Can handle noise effectively.
✔ Efficient for large datasets with spatial indexing (e.g., KD-trees, R-trees).

Disadvantages of DBSCAN

✘ Choosing optimal ε\varepsilon and minPts is not trivial.
✘ Struggles with varying density clusters.
✘ Sensitive to parameter selection.
✘ Not ideal for high-dimensional data due to distance measure limitations.

Kernel Density Estimation (KDE)

- KDE estimates the density of data points using kernel functions (e.g., Gaussian kernels).
- Computes a continuous density function instead of discrete clusters.
- Used in DENCLUE (Density-Based Clustering), which clusters based on density attractors.

Applications of Density-Based Clustering

→ Geospatial Data: Identifying geographic hotspots.
→ Anomaly Detection: Fraud detection, intrusion detection.
→ Image Segmentation: Grouping pixels into meaningful objects.
→ Astronomy: Detecting celestial structures in space.

Summary

- Density-based clustering groups points into dense regions.
- DBSCAN is widely used for its ability to detect noise and non-spherical clusters.
- KDE provides a continuous density estimation useful for clustering approaches like DENCLUE.
- Selecting optimal parameters is challenging but crucial for performance.

Model-Based Clustering

Overview

Model-based clustering assumes that **data is generated by an underlying probabilistic model**. Instead of assigning points to clusters based on distances or density, it estimates the **parameters of probability distributions** that best explain the data.

Key Characteristics

- **Assumes clusters follow specific probability distributions**.
- **Uses statistical models** to infer cluster assignments.
- **Can estimate the number of clusters automatically** using likelihood-based criteria.

- **Commonly applied in Gaussian Mixture Models (GMMs)**.

Gaussian Mixture Models (GMM)

A widely used model-based clustering method that assumes data is generated from a **mixture of Gaussian distributions**.

Algorithm: Expectation-Maximization (EM) for GMM

1. **Initialize parameters** (means, variances, mixing coefficients).
2. **Expectation (E-Step)**:
    - Compute the probability that each point belongs to each Gaussian component.
3. **Maximization (M-Step)**:
    - Update Gaussian parameters to maximize the likelihood of the data.
4. **Repeat steps 2-3 until convergence**.

Mathematical Formulation

The probability density function of a Gaussian Mixture Model is:

$$P(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$$

where:

- K = Number of clusters.
- π_k = Mixing coefficient of the kk-th Gaussian.
- $N(x|\mu_k, \Sigma_k)$ = Gaussian distribution with mean μ_k and covariance matrix Σ_k.

Advantages of GMM

✔ **Soft clustering**: Each point belongs to multiple clusters with different probabilities.
✔ **Can model elliptical clusters**, unlike K-Means which assumes spherical shapes.

✔ **More flexible than centroid-based clustering**.
✔ **Automatically determines cluster shape and spread**.

Disadvantages of GMM

✘ **Computationally expensive**, especially for high-dimensional data.
✘ **Requires assumption that data follows Gaussian distributions**.
✘ **Sensitive to initialization** (can converge to local optima).
✘ **Not robust to outliers**.

Applications of Model-Based Clustering

- **Genetics**: Identifying population subgroups.
- **Image Processing**: Object recognition and segmentation.
- **Finance**: Credit risk modeling.
- **Marketing**: Customer segmentation with probabilistic profiles.

Summary

→ Model-based clustering estimates probability distributions to define clusters.
→ GMMs assume data is generated from a mixture of Gaussians.
→ Expectation-Maximization (EM) is used to iteratively refine cluster assignments.
→ More flexible than K-Means but computationally more demanding.

Final Remarks

Summary of Clustering Methods

Clustering methods can be broadly categorized into:

- **Partitioning Methods** (e.g., K-Means) – Assign data points to K clusters.
- **Hierarchical Clustering** – Builds a nested structure of clusters.
- **Density-Based Clustering** (e.g., DBSCAN) – Identifies clusters as dense regions.
- **Model-Based Clustering** (e.g., GMM) – Assumes data is generated by probabilistic models.

Strengths and Weaknesses of Clustering Approaches

Method	Strengths	Weaknesses
K-Means	Fast, scalable, easy to interpret	Assumes spherical clusters, sensitive to K
Hierarchical Clustering	No need to define K, interpretable dendrogram	Computationally expensive
DBSCAN	Detects arbitrarily shaped clusters, handles noise	Struggles with varying density clusters
GMM	Flexible, probabilistic cluster assignments	Computationally expensive, assumes Gaussian distributions

Scalability Considerations

- → **K-Means and DBSCAN** scale well for large datasets.
- → **Hierarchical Clustering and GMM** become expensive with increasing data points.

- Advanced techniques like **MiniBatch K-Means** and **approximate DBSCAN** improve efficiency.

Research and Future Directions

- **Big Data Clustering**: Developing scalable algorithms for massive datasets.
- **High-Dimensional Clustering**: Addressing challenges with sparse data.
- **Deep Learning & Clustering**: Using neural networks for better feature representation.
- **Semi-Supervised Clustering**: Combining clustering with labeled data.

Applications of Clustering

Clustering is widely used in various domains:

- **Biology**: Gene expression analysis, species classification.
- **Finance**: Customer segmentation, fraud detection.
- **Image Processing**: Object detection, image segmentation.
- **Social Networks**: Community detection, recommendation systems.

Conclusion

Clustering is a fundamental unsupervised learning technique used for **pattern discovery** and **data structuring**. Choosing the right clustering method depends on **data characteristics**, **computational constraints**, and **the desired level of interpretability**.

Proximity measures

Similarity and Dissimilarity

Overview

Proximity measures define how alike or different two data objects are. These measures are categorized into similarity (how alike two objects are) and dissimilarity (how different two objects are).

Similarity

- ✓ A numerical measure indicating how alike two objects are.
- ✓ Higher similarity means objects are more alike.
- ✓ Often falls within the range [0,1].

Dissimilarity

- ✘ A numerical measure indicating how different two objects are.
- ✘ Lower dissimilarity means objects are more alike.
- ✘ The minimum dissimilarity is 0, while the upper limit varies.

Proximity is a general term that refers to either similarity or dissimilarity.

Similarity and Dissimilarity by Attribute Type

For two data objects with values pp and qq of a given attribute:

Attribute Type	Dissimilarity d(p,q)	Similarity s(p,q)
Nominal	$d = 0 \; if \; p = q \; (same \; value)$ $d = 1 \; if \; p \neq q \; (different \; value)$	$s = 1 \; if \; p = q$ $s = 0 \; if \; p \neq q$
Ordinal	$d = \frac{\lvert p - q \rvert}{V - 1}$ *(where V is the total number of possible values)*	$s = 1 \; \frac{\lvert p - q \rvert}{V - 1}$
Interval/Ratio	$d = \lvert p - q \rvert$	$s = \frac{1}{1 + \lvert p - q \rvert}$ $s = 1 - \frac{\lvert p - q \rvert - min(d)}{max(d) - min(d)}$

Where V is the number of possible values for an ordinal attribute.

Summary

- Similarity and dissimilarity define relationships between data objects.
- Proximity can refer to either similarity or dissimilarity.
- Different attribute types require different similarity/dissimilarity measures.
- These metrics are fundamental in clustering, classification, and information retrieval.

Distance Metrics

Overview

Distance metrics measure the dissimilarity between data objects in a multi-dimensional space. These measures are fundamental for clustering, classification, and retrieval tasks.

1. Euclidean Distance (L2 Norm)
Measures the straight-line distance between two points.

✓ Formula:

$$dist(p,q) = \sqrt{\sum_{d=1}^{D} (p_d - q_d)^2}$$

Used in K-Means, K-Nearest Neighbors (KNN), and SVMs.

→ Requires feature scaling if dimensions have different ranges.

2. Minkowski Distance (Lr Norm)

Generalization of Euclidean and Manhattan distances.

✓ Formula:

$$dist(p,q) = (\sum_{d=1}^{D} |p_d - q_d|^r)^{\frac{1}{r}}$$

Special cases:

- L1 Norm (Manhattan Distance): When r = 1
- L2 Norm (Euclidean Distance): When r = 2
- L∞ Norm (Chebyshev Distance): When r → ∞

3. Manhattan Distance (L1 Norm)

Computes the sum of absolute differences between coordinates.

✓ Formula:

$$dist(p,q) = \sum_{d=1}^{D} | p_d - q_d |$$

Suitable for high-dimensional and sparse data.

→ Used in grid-based applications and Lasso Regression.

4. Chebyshev Distance (L∞ Norm)

Measures the maximum absolute difference in any dimension.

✓ Formula:

$$dist(p,q) = \max_d |p_d - q_d|$$

→ Used in anomaly detection, chessboard distance, and industrial quality control.

5. Mahalanobis Distance

Accounts for correlations between variables by considering the covariance matrix.

✓ Formula:

$$dist_M(p,q) = \sqrt{(p-q)^T \Sigma^{-1} (p-q)}$$

Useful when features have different variances and correlations.

→ Applied in multivariate anomaly detection and classification.

Common Properties of a Distance Function

A valid distance metric satisfies:

1. **Positive definiteness:**

$$dist(p,q) \geq 0 \; and \; dist(p,q) = 0 \; if \; and \; only \; p = q$$

2. **Symmetry:**

$$dist(p,q) = dist(q,p)$$

3. **Triangle inequality:**

$$dist(p, q) \leq dist(p, r) + dist(r, q).$$

Summary

- Distance metrics measure dissimilarity between data objects.
- Euclidean and Manhattan distances are widely used but require scaling.
- Minkowski distance generalizes multiple distance functions.
- Mahalanobis distance considers feature correlations.
- The choice of distance metric impacts clustering, retrieval, and classification performance.

Similarity Measures

Overview

Similarity measures quantify how alike two data objects are. These metrics are fundamental in clustering, recommendation systems, text mining, and image retrieval.

1. Similarity Between Binary Vectors

Binary vectors represent categorical or presence/absence data. Two common similarity measures for binary data are:

Simple Matching Coefficient (SMC)

Measures the proportion of matching attributes in two binary vectors.

✓ Formula:

$$SMC = \frac{M_{00} + M_{11}}{M_{00} + M_{01} + M_{10} + M_{11}}$$

Where:

- M_{00} = Count of 0s in both vectors.

- M_{11} = Count of 1s in both vectors.
- M_{01}, M_{10} = Mismatched elements.

Jaccard Coefficient

Measures similarity by considering only attributes where at least one of the two vectors has a 1.

✓ Formula:

$$JC = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Often used in text mining, clustering, and social network analysis.

2. Cosine Similarity

Measures the cosine of the angle between two vectors.

→ Commonly used for text similarity in NLP applications.

✓ Formula:

$$cos(p, q) = \frac{p \cdot q}{||p||\,||q||}$$

✓ Example Applications:
- Document similarity in information retrieval.
- Collaborative filtering in recommendation systems.

3. Extended Jaccard Coefficient (Tanimoto Similarity)

Generalizes Jaccard similarity for continuous or count attributes.

✓ Formula:

$$T(p,q) = \frac{p \cdot q}{||p||^2 + ||q||^2 - p \cdot q}$$

Used in image processing, chemoinformatics, and pattern recognition.

Summary

- Similarity measures quantify how alike data points are.
- SMC and Jaccard Coefficient are used for binary vectors.
- Cosine Similarity is key in text analysis and recommendation systems.
- Extended Jaccard (Tanimoto) extends similarity for continuous attributes.
- Choosing the right similarity metric depends on data type and application.

Use Cases for Distance and Similarity Metrics

Overview

Different distance and similarity measures are useful in various applications, depending on data structure and the problem at hand.

1. Manhattan Distance (L1 Norm)

Use Cases

1. **Sparse High-Dimensional Data:** Effective when features are independent and not densely populated.
2. **Grid-Based Systems:** Applied in urban planning, robotics, and shortest path problems.
3. **Lasso Regression:** Helps with feature selection by shrinking coefficients of less important features to zero.

2. Chebyshev Distance (L∞ Norm)

Use Cases

1. **Anomaly Detection:** Efficient for detecting extreme deviations across multiple features.

2. **Chessboard Distance:** Models real-world movement where maximum single-step moves matter.
3. **Industrial Quality Control:** Applied in tolerance checking and defect detection.

3. Cosine Similarity

Use Cases

1. **Text Mining and Document Similarity:** Used in search engines and recommendation systems to detect similar content.
2. **Image Similarity:** Applied in image retrieval systems to match similar images.
3. **Recommendation Systems:** Common in collaborative filtering for user-item recommendations.

4. Jaccard Similarity

Use Cases

1. **Document Similarity in Information Retrieval:** Detects plagiarism and duplicate content.
2. **Image Processing:** Measures similarity in object recognition and segmentation.
3. **Clustering and Community Detection:** Used in social network analysis to identify communities based on shared elements.

Summary

- Manhattan Distance is effective for high-dimensional sparse data.
- Chebyshev Distance is ideal for anomaly detection and real-world movement models.
- Cosine Similarity is widely used in text analysis and recommendation systems.
- Jaccard Similarity is essential in document retrieval, image processing, and clustering.
- Choosing the right metric depends on the problem domain and data properties.

Regression

Introduction to Regression

Overview

Regression is a fundamental supervised learning technique used to predict continuous values. The goal is to find a mathematical model that best describes the relationship between input features (independent variables) and a numeric target variable (dependent variable).

Key Characteristics of Regression

- Supervised learning task: Uses labeled data.
- Target variable is numerical: Unlike classification, which predicts categories.
- Minimizes prediction error: The objective is to reduce the difference between predicted and actual values.

Applications of Regression

Regression is widely used in various domains:

- **Finance:** Predicting stock prices, risk analysis.
- **Healthcare:** Disease progression modeling, medical cost estimation.
- **Marketing:** Customer demand forecasting, price elasticity analysis.
- **Engineering:** Failure prediction, energy consumption forecasting.

Summary

- Regression predicts continuous values by modeling relationships between input variables.
- It minimizes error to improve prediction accuracy.
- Used in diverse fields including finance, healthcare, marketing, and engineering.
- It is the foundation for more advanced predictive modeling techniques.

Linear Regression

Overview

Linear regression models the relationship between a dependent variable (target) and one or more independent variables (features) using a linear equation. The goal is to find the best-fitting line that minimizes the prediction error.

Mathematical Model

For a dataset with N observations and D features:

- x_i represents the D-dimensional input features.
- y_i represents the target variable.
- w is a D-dimensional vector of coefficients.

The linear model is given by:

$$y_i = w^T x_i + b$$

where:

- w represents the weights (coefficients).
- b is the intercept.

Objective Function and Minimization

The most common approach to estimate ww and bb is Least Squares Minimization, which minimizes the Sum of Squared Errors (SSE):

$$O = \sum_{i=1}^{N} (w^T x_i + b - y_i)^2$$

Setting the gradient of O to zero leads to the normal equation:

$$w = (X^T X)^{-1} X^T y$$

where X is the matrix of input features and y is the response vector.

Interpretation of Coefficients

- ✓ **Positive coefficients:** Increase in the feature value increases the target variable.
- ✘ **Negative coefficients:** Increase in the feature value decreases the target variable.

Issues in Linear Regression

1. **Multicollinearity:** When independent variables are highly correlated, coefficient estimates become unstable.
2. **Heteroscedasticity:** Variance of residuals should remain constant; otherwise, predictions become unreliable.
3. **Assumption of Linearity:** If the true relationship is non-linear, performance suffers.

Quality of the Fit - R^2 Score

The coefficient of determination (R^2) measures how well the model explains variability in the target:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

where:

- → $SS_{res} = \Sigma(y_i - \widehat{y}_i)^2$ (sum of squared residuals).
- → $SS_{tot} = \Sigma(y_i - \overline{y}_i)^2$ (total sum of squares).
- → R^2 ranges from 0 to 1, where higher values indicate a better fit.

Summary

- Linear regression finds the best linear relationship between input features and the target.
- Minimizes the sum of squared errors to estimate coefficients.
- R^2 score quantifies how well the model explains target variability.
- Assumptions like linearity and homoscedasticity must be checked for validity.

Multiple Regression

Overview

Multiple regression extends linear regression to handle multiple independent variables. It models the relationship between a dependent variable and multiple predictors, improving accuracy when multiple factors influence the outcome.

Mathematical model

For a dataset with N observations and D features, the regression model is:

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_D x_{iD} + \epsilon_i$$

where:

- y_i = Target variable.
- x_{ij} = Feature values.
- w_j = Regression coefficients.
- ϵ_i = Error term.

Estimating Coefficients

The least squares approach minimizes the sum of squared errors:

$$O = \sum_{i=2}^{N} (y_i - \widehat{y_i})^2$$

The solution uses the normal equation:

$$w = (X^T X)^{-1} X^T y$$

Interpretation of Coefficients

Each coefficient w_j represents the change in y when the corresponding x_j increases by 1, keeping other variables constant.

✓ Positive w_j → Feature has a positive impact on the target.

✗ Negative w_j → Feature has a negative impact on the target.

Issues in Multiple Regression

1. **Multicollinearity**

   Occurs when independent variables are highly correlated.

   Causes unstable coefficient estimates.

   - **Solution:** Variance Inflation Factor (VIF) to detect and remove correlated features.
2. **Overfitting**
   Happens when the model learns noise instead of the actual trend.
   - **Solution:** Use regularization techniques (Ridge, Lasso).
3. **Heteroscedasticity**

The variance of residuals should remain constant.

- **Solution:** Apply log transformation or weighted regression.

Multiple Regression in Scikit-Learn

Example implementation:

```python
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import numpy as np

# Sample dataset
X = np.random.rand(100, 3)  # Three independent variables
y = 3 * X[:, 0] + 2 * X[:, 1] + 5 * X[:, 2] + np.random.randn(100)  # True model

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Coefficients
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

Summary

- Multiple regression models the relationship between a target variable and multiple predictors.
- The least squares method estimates the regression coefficients.
- Multicollinearity, overfitting, and heteroscedasticity must be addressed.
- Scikit-learn provides tools to implement multiple regression efficiently.

Polynomial Regression

Overview

Polynomial regression extends linear regression by introducing polynomial terms to model non-linear relationships between independent and dependent variables. Instead of fitting a straight line, it fits a polynomial curve to the data.

Mathematical Model

A polynomial regression of degree d is represented as:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_D x_i^D + \epsilon_i$$

where:

- y_i = Target variable.
- x_i = Feature values.
- w_j = Regression coefficients.
- d = Degree of the polynomial.
- ϵ_i = Error term.

When to Use Polynomial Regression

- ✓ When relationships between variables are non-linear but cannot be handled by simple linear regression.
- ✓ When adding polynomial features improves model performance without overfitting.
- ✓ When residual plots indicate a non-linear pattern.

Overfitting and Model Complexity

- Low-degree polynomial (Underfitting): Fails to capture the pattern in data.
- High-degree polynomial (Overfitting): Captures noise rather than trends.

- Optimal Degree: Can be selected using cross-validation or regularization (Ridge/Lasso).

Implementing Polynomial Regression in Scikit-Learn

Example implementation:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
import numpy as np

# Sample dataset
X = np.random.rand(100, 1) * 10  # One independent variable
y = 3 * X**2 + 2 * X + 5 + np.random.randn(100, 1)  # Quadratic relationship

# Polynomial Regression Model (degree = 2)
model = make_pipeline(PolynomialFeatures(2), LinearRegression())
model.fit(X, y)

# Predict values
y_pred = model.predict(X)
```

Summary

- Polynomial regression fits non-linear relationships by adding polynomial terms to the model.
- Higher-degree polynomials increase flexibility but risk overfitting.
- Selecting the right degree is crucial to balance bias and variance.
- Scikit-learn provides `PolynomialFeatures` to easily implement polynomial regression.

Overfitting and Regularization

Overview

Overfitting occurs when a model learns the noise in the training data rather than the underlying pattern, leading to poor generalization on unseen data. Regularization techniques help control overfitting by penalizing large coefficients, ensuring a more generalizable model.

Overfitting vs. Underfitting

- → **Underfitting:** The model is *too simple* and fails to capture patterns.
- → **Overfitting:** The model is *too complex* and memorizes training data, failing on new data.

Goal: Find the right complexity balance using validation techniques.

Regularization Techniques

1. L1 Regularization (Lasso Regression)

Adds the absolute sum of coefficients to the loss function:

$$J(w) = \Sigma (y_i - \widehat{y}_i)^2 + \lambda \Sigma |w_j|$$

Encourages sparse models by setting some coefficients to zero.

→ Useful for feature selection.

2. L2 Regularization (Ridge Regression)

Adds the sum of squared coefficients to the loss function:

$$J(w) = \Sigma (y_i - \widehat{y}_i)^2 + \lambda \Sigma w_j^2$$

Reduces the impact of irrelevant features without eliminating them.

→ Helps with multicollinearity.

3. Elastic Net (Combination of L1 & L2)

Combines Lasso and Ridge regularization:

$$J(w) = \Sigma (y_i - \widehat{y}_i)^2 + \lambda_1 \Sigma |w_j| + \lambda_2 \Sigma w_j^2$$

→ Useful when features are correlated.

Provides both feature selection (L1) and coefficient shrinkage (L2).

Choosing the Right Regularization Method

Method	Feature Selection	Handles Multicollinearity	Shrinks Coefficients
Lasso (L1)	Yes	No	Yes
Ridge (L2)	No	Yes	Yes
Elastic Net	Yes	Yes	Yes

Regularization in Scikit-Learn

- `Lasso()` for L1 regularization.
- `Ridge()` for L2 regularization.
- `ElasticNet()` for a combination of both.
- Hyperparameter tuning via cross-validation (`LassoCV()`, `RidgeCV()`).

Summary

- Overfitting occurs when a model is too complex and learns noise.
- Regularization techniques (Lasso, Ridge, Elastic Net) prevent overfitting.
- L1 (Lasso) selects features, L2 (Ridge) shrinks coefficients, Elastic Net does both.
- Scikit-learn provides tools to apply and tune regularization effectively.

Model Selection (Comparison of Regression Models)

Overview

Selecting the best regression model depends on various factors such as model complexity, interpretability, predictive accuracy, and computational efficiency. This section compares different regression techniques and their applications.

Comparison of Regression Models

Regression Model	Handles Non-Linearity	Feature Selection	Handles Multicollinearity	Risk of Overfitting
Linear Regression	No	No	No	High
Multiple Regression	No	No	No	High
Polynomial Regression	Yes	No	No	Very High
Lasso Regression (L1)	No	Yes	No	Medium
Ridge Regression (L2)	No	No	Yes	Medium
Elastic Net	No	Yes	Yes	Medium

Factors to Consider When Choosing a Model

1. **Interpretability vs. Accuracy**

    → *Linear models are* **easy to interpret** but may lack accuracy for complex relationships.

- → *Polynomial regression* **captures non-linearity** but is prone to overfitting.
- → Regularized models (Lasso, Ridge, Elastic Net) *provide better* **generalization.**

2. **Multicollinearity Handling**

   - → Ridge regression is preferred when **independent variables are highly correlated.**
   - → Lasso regression performs **feature selection,** reducing the impact of correlated variables.

3. **Feature Selection**

   - → Lasso regression sets some coefficients to zero, automatically selecting features.
   - → Elastic Net combines L1 and L2 penalties, balancing feature selection and multicollinearity handling.

4. **Computational Efficiency**

   - → Linear and Ridge regression are computationally efficient.
   - → Polynomial regression and Elastic Net require more resources.

Model Selection in Scikit-Learn

Scikit-Learn provides tools to compare and select the best regression model:

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.model_selection import cross_val_score

models = {
    'Linear': LinearRegression(),
    'Ridge': Ridge(alpha=1.0),
    'Lasso': Lasso(alpha=0.1),
    'Elastic Net': ElasticNet(alpha=0.1, l1_ratio=0.5)
}

for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring='r2')
    print(f"{name}: Mean R² = {scores.mean():.3f}")
```

Summary

- Model selection depends on the trade-off between accuracy, interpretability, and computational cost.
- Linear models are simple but prone to overfitting.
- Regularization (Lasso, Ridge, Elastic Net) improves generalization and feature selection.
- Polynomial regression captures non-linearity but requires careful tuning to avoid overfitting.
- Scikit-Learn provides tools for model comparison and selection.

Regression-Regularized Techniques

Lasso Regression

Overview

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a form of linear regression with L1 regularization, which shrinks some coefficients to exactly zero, effectively performing feature selection.

Function Objective

Lasso minimizes the sum of squared errors while adding an L1 penalty:

$$J(w) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{D} |w_j|$$

where:

- **First term:** Ordinary least squares loss (measures prediction error).
- **Second term:** L1 penalty, which encourages sparsity.
- **λ:** Regularization parameter controlling penalty strength.

Effects of L1 Regularization

- **Feature Selection:** Coefficients of less relevant features become exactly zero, effectively removing them from the model.
- **Sparse Solution:** Only the most important predictors remain.
- **Robust to Multicollinearity:** If features are highly correlated, Lasso picks one and shrinks the rest to zero.

Computational Complexity

Solving Lasso is more complex than Ridge because *the absolute value function is non-differentiable*.

→ Algorithms like Coordinate Descent or Least Angle Regression (LARS) are used to optimize Lasso efficiently.

Interpretation of Coefficients

- When **λ is small,** Lasso behaves like **ordinary least squares (OLS).**
- When **λ is large,** Lasso aggressively shrinks coefficients to zero, **reducing model complexity.**

Advantages of Lasso Regression

✓ Automatic feature selection (ideal for high-dimensional data).
✓ Helps prevent overfitting by eliminating irrelevant predictors.
✓ Simplifies models, making them more interpretable.
✓ Works well when many features are irrelevant or redundant.

Disadvantages of Lasso Regression

✗ Sensitive to feature correlation (selects only one feature from a group of correlated features).
✗ Performance degrades when the number of features is much larger than the number of observations.
✗ May over-penalize weak but useful predictors.

Applications of Lasso Regression

- Genetics: Selecting key genes from thousands of possible predictors.

- Finance: Identifying the most important economic indicators affecting market trends.
- Medical Research: Determining the most relevant factors influencing disease progression.
- Marketing Analytics: Identifying top features affecting customer conversion rates.

Summary

- Lasso Regression applies L1 regularization, which forces some coefficients to be exactly zero, performing feature selection.
- Helps with high-dimensional datasets, but struggles when features are highly correlated.
- Scikit-Learn provides Lasso via `Lasso()` and `LassoCV()` for automatic hyperparameter tuning.

Ridge Regression

Overview

Ridge Regression applies L2 regularization, adding a penalty term to shrink coefficients and reduce overfitting while keeping all features.

Function Objective

Ridge minimizes the sum of squared errors with an L2 penalty:

$$J(w) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{D} w_j^2$$

where:

- **First term:** Ordinary least squares loss (measures prediction error).
- **Second term:** L2 penalty, which shrinks coefficients but does not set them to zero.
- **λ:** Regularization parameter controlling shrinkage intensity.

Effects of L2 Regularization

- ✓ **Reduces the impact of less important features** without eliminating them.
- ✓ **Prevents overfitting** by controlling coefficient magnitude.
- ✓ *Better* suited for **handling multicollinearity** than Lasso.

Choosing the Regularization Parameter λ

- **Higher λ** → More shrinkage, simpler model, less overfitting.
- **Lower λ** → Less shrinkage, behaves like ordinary least squares.
- **Optimal λ** is found via cross-validation (`RidgeCV` in Scikit-Learn).

Differences Between Ridge and Lasso

Feature	Ridge (L2)	Lasso (L1)
Shrinks Coefficients	Yes	Yes
Sets Coefficients to Zero	No	Yes
Handles Multicollinearity	Yes	No
Feature Selection	No	Yes

Advantages of Ridge Regression

- ✓ Reduces overfitting while retaining all features.
- ✓ Handles multicollinearity better than Lasso.
- ✓ More stable coefficient estimates in high-dimensional data.

Disadvantages of Ridge Regression

- ✗ Does not perform feature selection (keeps all features with small coefficients).
- ✗ Sensitive to feature scaling (requires normalization of inputs).
- ✗ Choosing λ requires cross-validation.

Applications of Ridge Regression

- Finance: Predicting stock returns while managing multicollinearity.
- Genetics: Handling correlated genetic markers.
- Marketing: Sales forecasting with highly correlated variables.
- Healthcare: Medical cost prediction based on patient features.

Ridge Regression in Scikit-Learn

 Example implementation:

```
from sklearn.linear_model import RidgeCV
from sklearn.model_selection import cross_val_score

ridge = RidgeCV(alphas=[0.1, 1.0, 10.0], store_cv_values=True)
ridge.fit(X_train, y_train)
print("Optimal λ:", ridge.alpha_)
```

Summary

- Ridge Regression applies L2 regularization, shrinking coefficients but keeping all features.
- Best for handling multicollinearity while preventing overfitting.
- Larger λ increases regularization, reducing coefficient magnitudes.
- Scikit-learn provides Ridge via `Ridge()` and `RidgeCV()` for hyperparameter tuning.

Elastic Net Regression

Overview

Elastic Net Regression combines L1 (Lasso) and L2 (Ridge) regularization, balancing feature selection and coefficient shrinkage.

Function Objective

Elastic Net minimizes the sum of squared errors with both L1 and L2 penalties:

$$J(w) = \Sigma (y_i - \hat{y}_i)^2 + \lambda_1 \Sigma |w_j| + \lambda_2 \Sigma w_j^2$$

where:

- **L1 penalty (λ_1):** Performs feature selection by setting some coefficients to zero.
- **L2 penalty (λ_2):** Shrinks coefficient magnitudes, reducing overfitting.
- Elastic Net balances both effects through a mixing parameter α\alpha where:
  - → α = 1 → Pure Lasso.
  - → α = 0 → Pure Ridge.

When to Use Elastic Net

✓ When **features are highly correlated** (Lasso struggles with this issue).
✓ When **feature selection is needed but Ridge regression is too weak.**
✓ For **high-dimensional data** where Lasso selects too few features.

Differences Between Ridge, Lasso, and Elastic Net

Feature	Ridge (L2)	Lasso (L1)	Elastic Net
Shrinks Coefficients	Yes	Yes	Yes
Sets Coefficients to Zero	No	Yes	Yes
Handles Multicollinearity	Yes	No	Yes
Feature Selection	No	Yes	Yes

Advantages of Elastic Net

✓ Combines L1 and L2 penalties for balanced regularization.
✓ Handles multicollinearity better than Lasso.
✓ Useful when there are many correlated features.
✓ More stable feature selection compared to Lasso alone.

Disadvantages of Elastic Net

- ✗ Requires tuning of two hyperparameters (λ_1, λ_2).
- ✗ Not as interpretable as Lasso for feature selection.
- ✗ More computationally expensive than Ridge or Lasso alone.

Applications of Elastic Net Regression

- Genetics: Selecting relevant genes in high-dimensional datasets.
- Finance: Predicting stock trends when economic indicators are highly correlated.
- Medical Research: Identifying key biomarkers while avoiding overfitting.
- Marketing Analytics: Handling datasets with redundant predictor variables.

Elastic Net in Scikit-Learn

 Example implementation:

```
from sklearn.linear_model import ElasticNetCV

elastic_net = ElasticNetCV(l1_ratio=[0.1, 0.5, 0.9], alphas=[0.1, 1.0, 10.0], cv=5)
elastic_net.fit(X_train, y_train)
print("Optimal λ:", elastic_net.alpha_)
print("Optimal L1 ratio:", elastic_net.l1_ratio_)
```

Summary

- Elastic Net combines L1 and L2 regularization, balancing feature selection and coefficient shrinkage.
- More robust than Lasso when dealing with highly correlated features.
- The mixing parameter α controls the balance between L1 and L2.
- Scikit-learn provides Elastic Net via `ElasticNet()` and `ElasticNetCV()` for hyperparameter tuning.

Comparison of Lasso, Ridge, and Elastic Net

Feature	Ridge (L2)	Lasso (L1)	Elastic Net
Shrinks Coefficients	Yes	Yes	Yes
Sets Coefficients to Zero	No	Yes	Yes
Handles Multicollinearity	Yes	No	Yes
Feature Selection	No	Yes	Yes
Works Well with Many Features	Yes	No	Yes
Computational Cost	Low	Low	Higher

When to Use Each Method

Scenario	Best Choice
Features are highly correlated	**Ridge**
Need automatic feature selection	**Lasso**
Need balance between Ridge and Lasso	**Elastic Net**
Many irrelevant features	**Lasso**
Want to retain all features with small effects	**Ridge**
Lasso selects too few features	**Elastic Net**