Module-2

Chapter – 01 - Understanding Data – 2

Bivariate Data and Multivariate Data

Bivariate Data

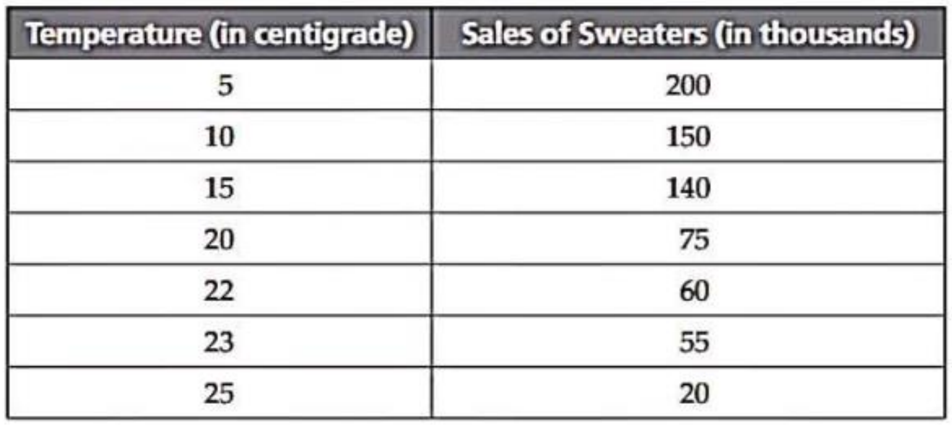
Bivariate data involves two variables, and the goal of bivariate analysis is to explore the relationship between them.

This relationship can help in comparisons, identifying causes, and further exploration of the data.

Bivariate Data involves two variables. Bivariate data deals with causes of relationships. The aim is to find relationships among data.

Consider the following Table 2.3, with data of the temperature in a shop and sales of sweaters.

Table 2.3: Temperature in a Shop and Sales Data



Temperature (in centigrade)	Sales of Sweaters (in thousands)
5	200
10	150
15	140
20	75
22	60
23	55
25	20

Scatter Plot

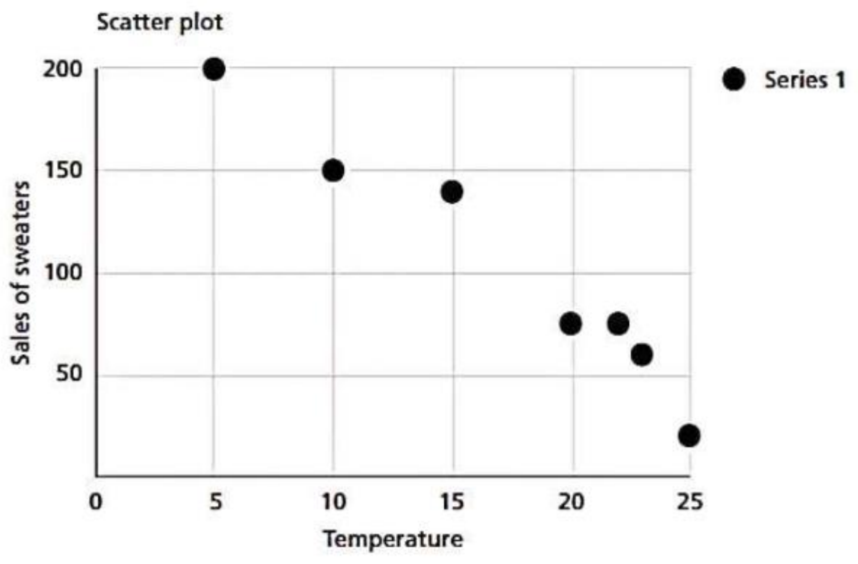


Figure 2.11: Scatter Plot

Line graphs are similar to scatter plots. The Line Chart for sales data is shown in Figure 2.12.

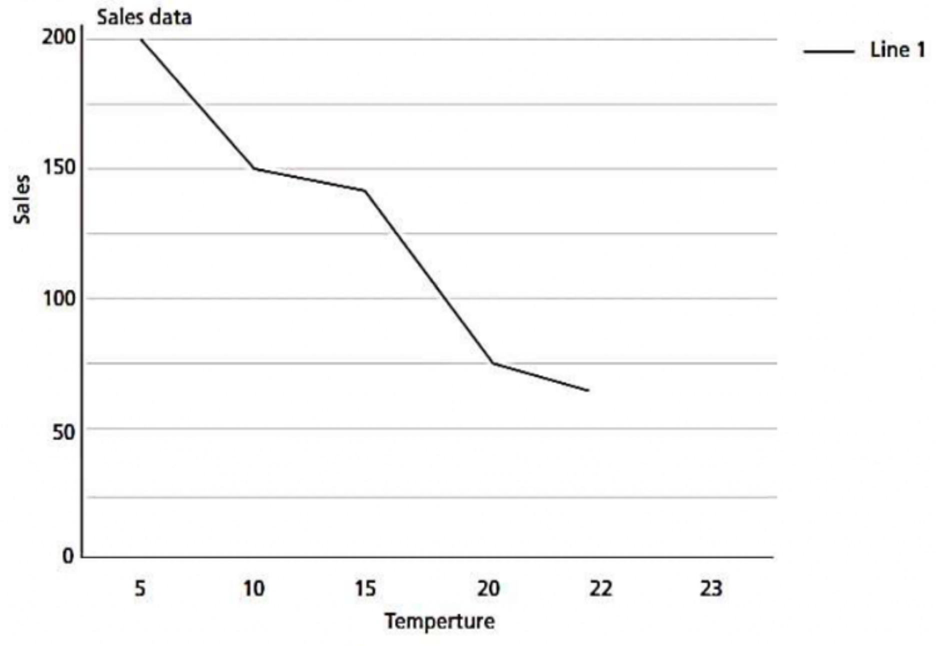


Figure 2.12: Line Chart

A scatter plot is a useful graphical method for visualizing bivariate data.

It is particularly effective for illustrating the relationship between two variables.

The key features of a scatter plot are:

- **Strength**: Indicates how closely the data points fit a pattern or trend.
- **Shape**: Helps in identifying the type of relationship (linear, quadratic, etc.).
- **Direction**: Shows whether the relationship is positive, negative, or neutral.
- **Outliers**: Helps identify any points that deviate significantly from the trend.

Scatter plots are often used in the exploratory phase of data analysis before calculating correlation coefficients or fitting regression models.

Bivariate Statistics

There are various statistical measures to describe the relationship between two variables.

Two important bivariate statistics are **Covariance** and **Correlation**.

Covariance

Covariance measures the joint variability of two random variables. It tells you whether an increase in one variable results in an increase or decrease in the other variable. Mathematically, the covariance between two variables X and Y is defined as:

$$\mathrm{COV}(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - E(X))(y_i - E(Y))$$

Where:

- x_i and y_i are individual data points from variables X and Y.
- $E(X)$ and $E(Y)$ are the means of X and Y, respectively.
- N is the number of data points.

Covariance values:

- **Positive covariance**: As one variable increases, the other variable also increases.
- **Negative covariance**: As one variable increases, the other variable decreases.
- **Zero covariance**: No linear relationship between the variables.

Covariance is symmetric, meaning $\text{COV}(X, Y) = \text{COV}(Y, X)$.

Correlation

While covariance measures the direction of the relationship, **correlation** quantifies the strength of the relationship between two variables.

The most common measure of correlation is the **Pearson correlation coefficient**:

$$r = \frac{\text{COV}(X, Y)}{\sigma_X \sigma_Y}$$

Where:

- $\text{COV}(X, Y)$ is the covariance between X and Y.
- σ_X and σ_Y are the standard deviations of X and Y, respectively.

The value of r ranges from -1 to +1:

- **+1**: Perfect positive correlation.
- **-1**: Perfect negative correlation.
- **0**: No linear relationship between X and Y.

Unlike covariance, correlation is **dimensionless**, meaning it is not affected by the units of the variables.

Multivariate Statistics

Multivariate data refers to data that involves more than two variables, and in machine learning, most datasets are multivariate.

The goal of multivariate analysis is to understand relationships among multiple variables simultaneously.

This can involve multiple dependent (response) variables, and is often used for analyzing more complex data scenarios.

Multivariate analysis techniques include:

- **Regression Analysis**
- **Principal Component Analysis (PCA)**
- **Path Analysis**

The **mean vector** is used to represent the mean of multiple variables, and the **covariance matrix** represents the variance and relationships among all variables.

The **mean vector** is also known as the **centroid**, while the **covariance matrix** is also referred to as the **dispersion matrix**.

Multivariate Analysis Techniques

Regression Analysis:

Used to model the relationship between multiple independent variables and a dependent variable.

Factor Analysis:

A statistical method used to identify underlying relationships between observed variables.

Multivariate Analysis of Variance (MANOVA):

Extends ANOVA to analyze multiple dependent variables simultaneously.

Visualization Techniques for Multivariate Data

Heatmap

A **heatmap** is a graphical representation of a 2D matrix where values are represented by colors. In a heatmap:

- Darker colors indicate larger values.
- Lighter colors indicate smaller values.

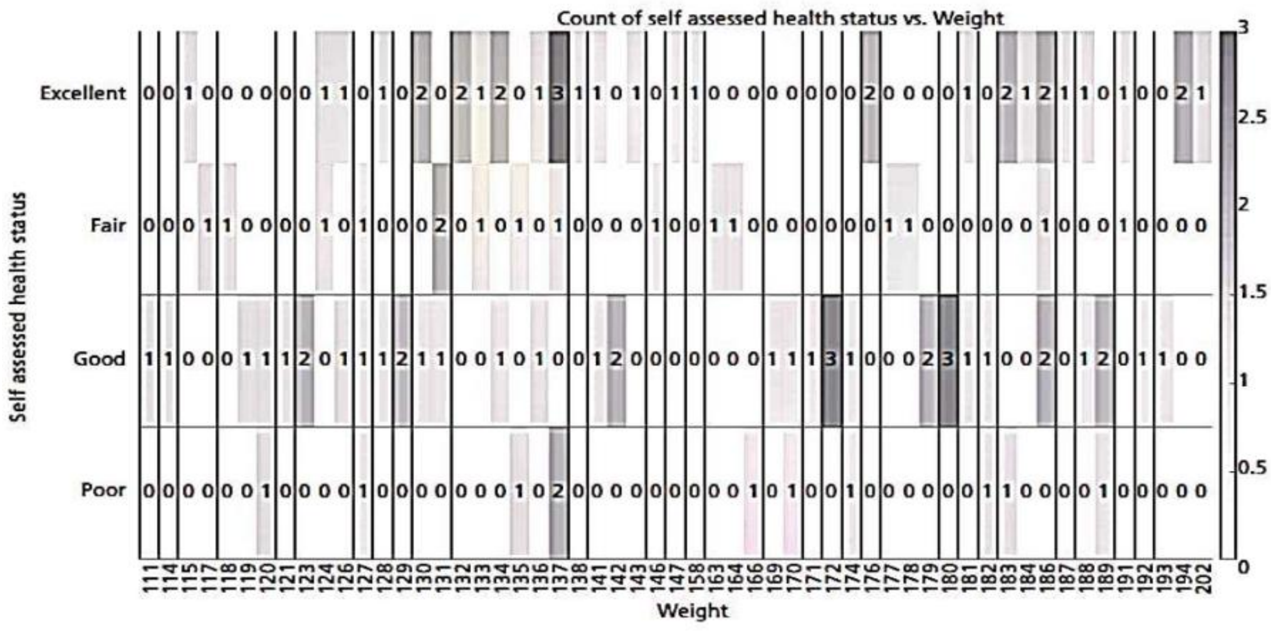


Figure 2.13: Heatmap for Patient Data

Applications:

Heatmaps are useful for visualizing complex data like traffic patterns or patient health data, where you can easily identify regions of higher or lower values.

Example:

In vehicle traffic data, regions with heavy traffic are highlighted with dark colors, making it easy to spot problem areas.

Pairplot (or Scatter Matrix)

A **pairplot** (or **scatter matrix**) is a matrix of scatter plots that shows relationships between every pair of variables in a multivariate dataset.

This method allows you to visually examine correlations or relationships between variables.

A random matrix of three columns is chosen and the relationships of the columns is plotted as a pairplot (or scattermatrix) as shown below in Figure 2.14.

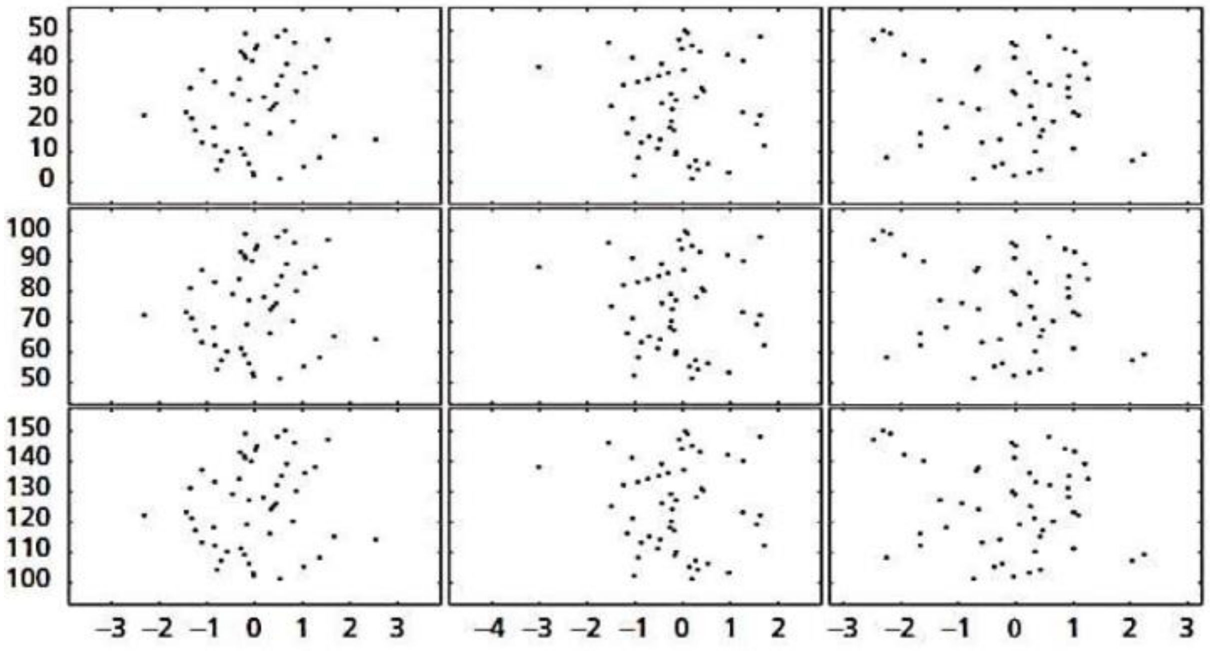


Figure 2.14: Pairplot for Random Data

- **Visual Layout**: Each scatter plot in the matrix shows the relationship between two variables.
- **Usefulness**: By examining the pairplot, you can easily identify patterns, correlations, or clusters among the variables.

Essential Mathematics for Multivariate Data

In the realm of machine learning and multivariate data analysis, several mathematical concepts are foundational.

These include concepts from **Linear Algebra**, **Statistics**, **Probability**, and **Optimization**. Below is an overview of essential mathematical tools that are necessary for understanding and working with multivariate data.

Linear Algebra

Linear algebra is crucial in machine learning as it provides the tools for dealing with data in the form of vectors and matrices. Here's a breakdown of important topics:

- **Vectors**: A vector is an ordered list of numbers. It can represent data points or features of an observation in a multivariate dataset.
    - **Dot product** and **cross product** are used to compute projections and angles between vectors.
- **Matrices**: A matrix is a 2D array of numbers. In machine learning, matrices often represent data where rows are instances and columns are features.
    - **Matrix multiplication** allows the transformation of data and is used in various algorithms like linear regression, neural networks, and more.
- **Eigenvalues and Eigenvectors**: These are important for dimensionality reduction techniques such as **Principal Component Analysis (PCA)**. They are used to transform data into a new basis that captures the most variance.
- **Determinants and Inverses**: The determinant of a matrix tells us if the matrix is invertible (non-singular). The inverse of a matrix is used to solve linear systems of equations.
- **Singular Value Decomposition (SVD)**: This is a factorization method used in PCA and other dimensionality reduction techniques to decompose a matrix into singular values and vectors.

Statistics

Statistics is key to understanding the relationships between different variables in multivariate data. Key concepts include:

- **Mean and Variance**: Measures of central tendency (mean) and spread (variance) are essential to understanding the distribution of each variable.
- **Covariance**: Covariance measures the relationship between two variables. A positive covariance indicates that as one variable increases, the other tends to increase.
- **Correlation**: Correlation is a normalized measure of covariance that indicates the strength and direction of the relationship between two variables.
- **Multivariate Normal Distribution**: Many machine learning algorithms assume that the data follows a multivariate normal distribution, which extends the idea of normal distribution to more than one variable.
- **Principal Component Analysis (PCA)**: PCA is used to reduce the dimensionality of the dataset while retaining as much variance as possible. It uses eigenvectors and eigenvalues to identify the principal components.

Probability

Probability theory underpins the concept of uncertainty, which is inherent in real-world data:

- **Random Variables**: A random variable represents a quantity whose value is subject to chance. In multivariate data, we deal with vectors of random variables.
- **Probability Distributions**: These describe the likelihood of various outcomes. Common distributions in machine learning include the normal distribution and the multinomial distribution.

- **Bayes' Theorem**: This theorem describes the probability of an event, based on prior knowledge of related events. It's fundamental to algorithms like **Naive Bayes** and **Bayesian Inference**.
- **Markov Chains**: These are used for modeling systems that undergo transitions from one state to another with a certain probability, without memory of previous states.

Optimization

Optimization is key to finding the best model for multivariate data. Many machine learning algorithms are formulated as optimization problems.

- **Gradient Descent**: An iterative optimization algorithm used to minimize a cost function (such as in linear regression or neural networks).
- **Convex Optimization**: Involves minimizing convex functions, and plays a significant role in machine learning, as many cost functions are convex.
- **Lagrange Multipliers**: Used for optimizing functions subject to constraints, which is often seen in constrained optimization problems in machine learning.

Multivariate Analysis

- **Multivariate Regression**: This is the extension of linear regression to predict multiple dependent variables using a set of independent variables.
- **Multivariate Analysis of Variance (MANOVA)**: An extension of ANOVA used when there are two or more dependent variables. It tests for differences between groups.
- **Factor Analysis**: A method for identifying the underlying relationships between observed variables. It's often used in exploratory data analysis.

Graphical Techniques for Multivariate Data

- **Scatter Plots**: A scatter plot can be used to visualize the relationship between two variables. For multivariate data, **pair plots** or **scatter matrices** are used to examine the relationships between all pairs of variables.
- **Heatmaps**: Used to visualize correlation matrices or covariance matrices, where color intensity represents the strength of the relationship.

Multivariate Data Models

- **Multivariate Normal Distribution**: A generalization of the univariate normal distribution to multiple variables, frequently assumed in multivariate statistical analysis.
- **Multivariate Linear Models**: Models such as multiple regression, where multiple independent variables are used to predict a set of dependent variables.

Dimensionality Reduction

Dimensionality reduction is used to reduce the number of variables in a dataset while maintaining the essential information:

- **Principal Component Analysis (PCA)**: A technique that reduces the dimensionality of the dataset by projecting the data onto a set of orthogonal axes (principal components) that explain the most variance.
- **t-SNE**: A technique for dimensionality reduction that is well-suited for visualizing high-dimensional data in 2D or 3D space.

Feature Engineering and Dimensionality Reduction Techniques

Feature engineering and dimensionality reduction are critical steps in machine learning workflows.

They ensure that models are not only accurate but also efficient, interpretable, and scalable.

1. Feature Engineering

Feature engineering involves creating, modifying, or selecting features (variables) from raw data to improve the performance of machine learning models.

Techniques in Feature Engineering

1. **Feature Creation**
   - **Polynomial Features:** Adding higher-order terms (e.g., x^2, x^3) to model non-linear relationships.
   - **Interaction Features**: Creating features by combining two or more variables (e.g., $x_1 \times x_2$).
   - **Domain-Specific Features:** Designing features based on domain knowledge (e.g., calculating BMI from height and weight).
2. **Feature Transformation**
   - **Normalization**: Scaling values to a specific range, typically [0,1].
   - **Standardization**: Transforming features to have a mean of 0 and a standard deviation of 1.
   - **Log Transformation**: Reducing the impact of large values by applying the log function.
   - **Power Transformation**: Stabilizing variance by applying functions like square root or exponential transformations.
3. **Handling Missing Values**
   - **Imputation**: Filling missing values with statistical measures (mean, median, mode) or predictions from models.
   - **Dropping Features or Rows**: Removing features or samples with excessive missing data.

4. **Encoding Categorical Features**
    - **Label Encoding**: Assigning numerical values to categories.
    - **One-Hot Encoding**: Creating binary columns for each category.
    - **Target Encoding**: Replacing categories with the mean of the target variable.
5. **Feature Selection**
    - **Filter Methods**: Using statistical tests (e.g., correlation, chi-square) to select features.
    - **Wrapper Methods**: Selecting features based on the performance of a model (e.g., recursive feature elimination).
    - **Embedded Methods**: Feature selection integrated into model training (e.g., regularization methods like LASSO).

Dimensionality Reduction

Dimensionality reduction aims to reduce the number of features while preserving as much relevant information as possible.

It helps combat issues like **overfitting**, **high computational costs**, and the **curse of dimensionality**.

Techniques for Dimensionality Reduction

1. **Principal Component Analysis (PCA)**
    - **Purpose**: Identifies directions (principal components) in the data that explain the maximum variance.
    - Projects data onto a new coordinate system where each axis represents a principal component.
    - Captures the most variance in the first few components.

    Applications: Commonly used in image compression, gene expression analysis, and exploratory data analysis.

2. **Linear Discriminant Analysis (LDA)**
   - **Purpose**: Similar to PCA but focuses on maximizing class separability in supervised learning tasks.
   - Projects data onto a lower-dimensional space while maintaining class distinction.

   Applications: Often used in classification problems.

3. **t-Distributed Stochastic Neighbor Embedding (t-SNE)**
   - **Purpose**: Reduces high-dimensional data to 2D or 3D for visualization.
   - Preserves the local structure of the data while sacrificing global structure.

   Applications: Useful for visualizing clusters in high-dimensional data like embeddings.

4. **Autoencoders (Deep Learning-Based Reduction)**
   - **Purpose**: Learns a compressed representation of the data using neural networks.
   - The encoder compresses the data, and the decoder reconstructs it.
   - The bottleneck layer represents the reduced dimensions.

   Applications: Image compression, anomaly detection, and generative models.

5. **Feature Agglomeration**
   - **Purpose**: Groups features with similar characteristics (hierarchical clustering for features).
   - Combines redundant features into a single representative feature.

   Applications: Useful for datasets with many correlated features.

6. **Independent Component Analysis (ICA)**
    - **Purpose**: Decomposes data into statistically independent components.
    - Useful for signals with non-Gaussian distributions.

      Applications: Signal processing, such as separating audio signals in the "cocktail party problem."

7. **Factor Analysis**
    - **Purpose**: Identifies underlying latent variables (factors) that explain observed variables.
    - Assumes that observed data is influenced by a smaller number of unobservable factors.

      Applications: Psychometrics, finance, and social sciences.

8. **Backward Feature Elimination**
    - **Purpose**: Iteratively removes features that have the least impact on the target variable.
    - Uses a trained model's performance as the criterion.

      Applications: Effective for small datasets where computational cost isn't a concern.

Combining Feature Engineering and Dimensionality Reduction

Pipeline Integration:

Many machine learning frameworks (e.g., scikit-learn) support building pipelines where feature engineering and dimensionality reduction steps are automated.

Hybrid Methods:

For example:

- Combine **PCA** with **feature selection** to reduce noise and retain relevant features.
- Use **autoencoders** to generate compact features, then apply supervised learning techniques.

Applications

Text Data:

- Use **TF-IDF** for feature creation and **Latent Semantic Analysis (LSA)** for dimensionality reduction.

Image Data:

- Apply **Convolutional Autoencoders** or **PCA** for reducing pixel-based data dimensions.

Genomic Data:

- Use **PCA** or **t-SNE** to visualize high-dimensional gene expression data.

Sensor Data:

- Combine **Fourier transforms** for feature extraction and **PCA** for dimensionality reduction.

Best Practices

Understand Data: Always begin with exploratory data analysis (EDA) to understand feature importance and relationships.

Domain Knowledge: Incorporate domain expertise to create meaningful features.

Avoid Over-Reduction: Ensure that dimensionality reduction techniques retain sufficient information to build an accurate model.

Evaluate: Continuously evaluate feature engineering and dimensionality reduction using cross-validation.

Chapter – 02

Basic Learning Theory

Design of Learning System

A learning system is a computational system that uses algorithms to learn from data or experiences to improve its performance over time.

The design of such systems focuses on the following essential steps:

Choosing a Training Experience

The first step in building a learning system is selecting the type of training experience it will use to learn. This involves determining the source of data and how it will be used.

Types of Training Experience:

Direct Experience:

- The system is explicitly provided with examples of board states and their correct moves.
- Example: In a chess game, the system is given specific board states and the optimal moves for those states.

Indirect Experience:

- Instead of explicit guidance, the system is provided with sequences of moves and their results.
- Example: The system observes the outcome (win or loss) of different move sequences and learns to optimize its strategy.

Supervised vs. Unsupervised Training:

In **supervised training**, a supervisor labels all valid moves for a given board state.

In the absence of a supervisor, the system uses self-play or exploration to learn. For example, a chess agent can play games against itself and identify successful moves.

Training Data Distribution:

- For reliable performance, training samples must cover a wide range of scenarios.
- If the training data and testing data have similar distributions, the system's performance will be better.

Determining the Target Function

The target function represents the knowledge the system needs to learn.

It specifies the goal of the learning system and what it is trying to predict or optimize.

- **Direct Experience:**
  - The system evaluates whether a specific move M is a "good move" or not.
  - The target function can be expressed as:
  $$B \to M$$
  where B represents the current board state, and M represents the chosen move. The system selects the best move M based on this evaluation.
- **Indirect Experience:**
  - The system evaluates all legal moves and assigns a score to each.
  - The move with the highest score is chosen for execution:
  $$\text{Score}(B, M) \to \max$$

Representation of the Target Function

Once the target function is defined, the next step is deciding how to represent it. The representation depends on the complexity of the problem and the available computational resources.

Common Representations:

Lookup Tables:

- Used for simple problems where all possible states and actions can be enumerated.
- Example: A small chessboard with a limited number of moves.

Mathematical Functions:

- Represented using equations or models (e.g., linear regression or polynomial equations).

Machine Learning Models:

- For complex systems, models like neural networks, decision trees, or support vector machines are used to approximate the target function.
- Example: Using a neural network to predict the best chess moves based on board states.

Function Approximation

In most real-world problems, the target function is too complex to be represented exactly. Instead, an approximation of the target function is learned.

Approaches to Approximation:

Parametric Models:

Models with a fixed number of parameters (e.g., linear regression, neural networks).

Non-Parametric Models:

Models that adapt their complexity to the amount of data (e.g., k-nearest neighbors, decision trees).

Learning Algorithms:

- Algorithms like gradient descent, reinforcement learning, or evolutionary algorithms are used to optimize the parameters of the function.
- Example: In a chess game, reinforcement learning allows the agent to learn by trial and error, optimizing its strategy over time.

Practical Example: Designing a Chess Learning System

Training Experience:

Use a combination of self-play (indirect experience) and historical game data (direct experience).

Target Function:

Define the target function as selecting the best move M given the board state B:

$$f(B) \to M$$

Representation of the Target Function:

Use a deep neural network to represent the target function, where inputs are board states and outputs are move probabilities.

Function Approximation:

Train the neural network using reinforcement learning, with rewards based on the outcome of games played by the system.

Introduction to Concept of Learning

Concept learning is a strategy in machine learning that involves acquiring abstract knowledge or inferring general concepts from the given training data.

It enables the learner to generalize from specific training examples and classify objects or instances based on common, relevant features.

What is Concept Learning?

Concept learning is the process of **abstraction and generalization** from data, where:

- The learner identifies **common features** shared by positive examples.
- It uses these features to classify new instances into categories.

It involves:

- **Comparing and contrasting categories** by analyzing positive and negative examples.
- Simplifying observations from training data into a model or hypothesis.
- Applying this model to classify future data.

This process is also known as **learning from experience**.

Features of Concept Learning

Categorization:

- Concept learning enables classification of objects based on a set of **relevant features**.

- For example, humans classify animals like elephants, cats, or dogs based on specific distinguishing features.

Boolean-Valued Function:

- Each concept or category learned is represented as a Boolean function that returns true or false:
    - True for positive examples that belong to the category.
    - False for negative examples that do not belong to the category.

Example:

- Humans categorize animals by recognizing features such as:
    - Size, shape, color, and behavior.
- For example, to identify an elephant:
    - Large size, trunk, tusks, and big ears are the specific features.

Formal Definition of Concept Learning

Concept learning is the process of **inferring a Boolean-valued function** by processing training examples.

The goal is to:

1. Identify a set of **specific or common features**.
2. Use these features to define a **target concept** for classifying objects.

Components of Concept Learning

Input:

- A labeled **training dataset** consisting of:
    - Positive examples: Instances that belong to the target concept.

  - Negative examples: Instances that do not belong to the target concept.
- The learner uses this past experience to train the model.

Output:

- The **Target Concept** or **Target Function** f(x):
  - A function f(x) maps input x to output y.
  - The output is used to determine the relevant features for classification.
- Example: Identifying an elephant requires a specific set of features such as "has a trunk" and "has tusks."

Testing:

- New instances are provided to **test the learned model**.
- The system classifies these new instances based on the hypothesis derived during training.

Process of Concept Learning

Training:

- The learner observes a set of labeled examples (positive and negative instances).
- It identifies common, relevant features from the positive examples and contrasts them with negative examples.

Hypothesis Formation:

- The system generates a hypothesis to represent the target concept.
- Example: "An elephant has a trunk and tusks" could be the hypothesis to classify an elephant.

Generalization:

- The hypothesis is generalized to classify new instances correctly.

Testing and Validation:

- The learned model is tested on unseen data to evaluate its performance.

Example: Concept Learning for Animals

Input: Training dataset of animals with labeled features.

- Positive examples: Animals labeled as "elephants."
- Negative examples: Animals not labeled as "elephants."

Output: Target concept for an elephant, e.g., "has a trunk," "has tusks," and "large size."

Testing: New animal instances are classified based on the learned concept.

Applications of Concept Learning

1. **Natural Language Processing**: Categorizing words or sentences based on grammatical or semantic features.
2. **Image Recognition**: Identifying objects or patterns in images.
3. **Recommendation Systems**: Classifying products or services to provide personalized recommendations.
4. **Medical Diagnosis**: Identifying diseases based on symptoms and medical test results.

Modelling in Machine Learning

A machine learning model abstracts a training dataset and makes predictions on unseen data.

Training: Involves feeding training data into a machine learning algorithm, tuning parameters, and generating a predictive model.

Goals: Selecting the right model, training effectively, reducing training time, and achieving high performance on unseen data.

Types of Parameters:

Model Parameters: Learnable directly from training data (e.g., regression coefficients, decision tree splits, neural network weights).

Hyperparameters: Cannot be learned directly and must be set (e.g., regularization strength, number of trees in random forests).

Evaluation and Error Metrics

Dataset Splitting:

- Training dataset: Used to train the model.
- Test dataset: Used to evaluate the model's ability to generalize.

Error Types:

- **Training Error (In-sample Error)**: Error when the model is tested on training data.
- **Test Error (Out-of-sample Error)**: Error when predicting on unseen test data.

Loss Function: Measures prediction error. Example: **Mean Squared Error (MSE)**—a smaller value indicates higher accuracy.

Steps in Machine Learning Process

Algorithm Selection: Choose a model suitable for the problem and dataset.

Training: Train the selected algorithm on the dataset.

Tuning: Adjust parameters to improve accuracy.

Evaluation: Validate the model using test data.

Model Selection and Evaluation

Challenges:

Balancing **performance** (accuracy) and **complexity** (overfitting or underfitting).

Approaches:

1. Resampling methods like splitting datasets or cross-validation.
2. Calculating accuracy or error metrics.
3. Probabilistic frameworks for scoring model performance.

Resampling Methods

Random Train/Test Splits: Randomly split the data for training and testing.

Cross-Validation: Tune models by splitting data into folds:

- **K-fold Cross-Validation**: Split data into k parts, train on k-1 folds, and test on the remaining fold.

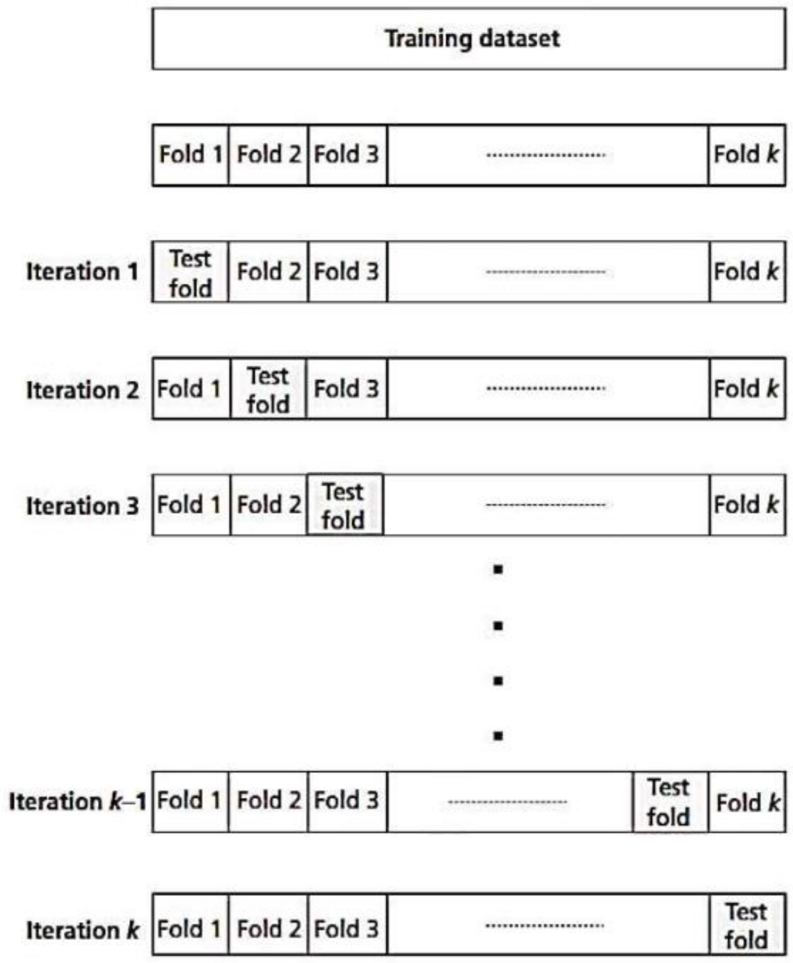

Training dataset
Fold 1 | Fold 2 | Fold 3 | .... | Fold k
Iteration 1 | Test fold | Fold 2 | Fold 3 | .... | Fold k
Iteration 2 | Fold 1 | Test fold | Fold 3 | .... | Fold k
Iteration 3 | Fold 1 | Fold 2 | Test fold | .... | Fold k
Iteration k–1 | Fold 1 | Fold 2 | Fold 3 | .... | Test fold | Fold k
Iteration k | Fold 1 | Fold 2 | Fold 3 | .... | Test fold

Figure 3.4: Illustration of *K*-fold Cross-Validation

Algorithm 3.4: *K*-fold Cross Validation

```
Input: Dataset with 'n' instances
       'k' – Number of folds
   1. Repeat the following steps for 'k' times with a different 'hold-out' fold for evaluation:
      •  Split the dataset into k equal folds.
      •  Train the model on (k – 1) folds.
      •  Evaluate it on the 1 remaining "hold-out" fold.
   2. Compute average of the performance across all k hold-out folds.
```

- **Stratified K-fold**: Ensures each fold contains a proportionate distribution of class labels.

- **Leave-One-Out Cross-Validation (LOOCV)**: Train on all data except one instance; repeat for every instance.

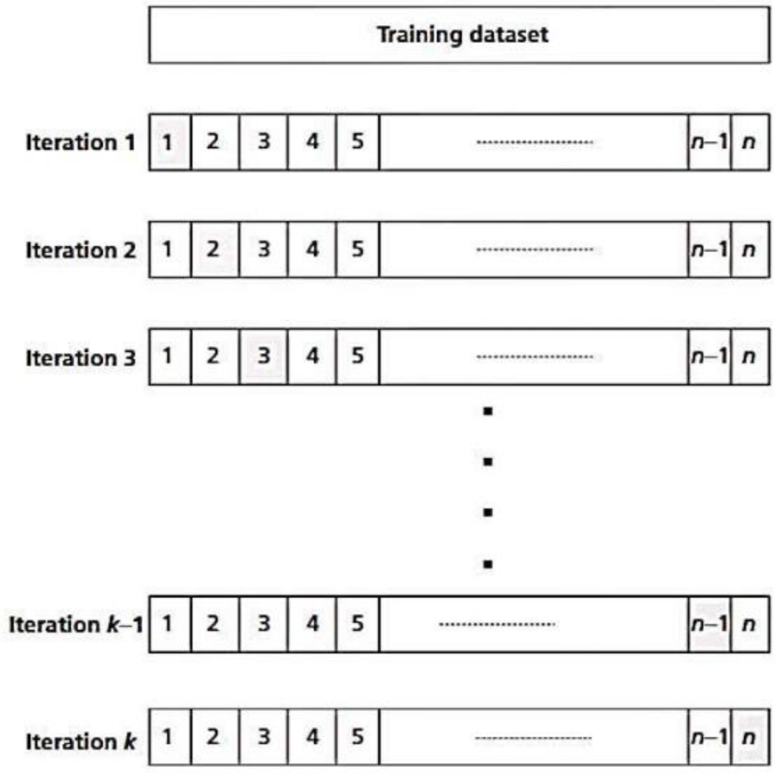
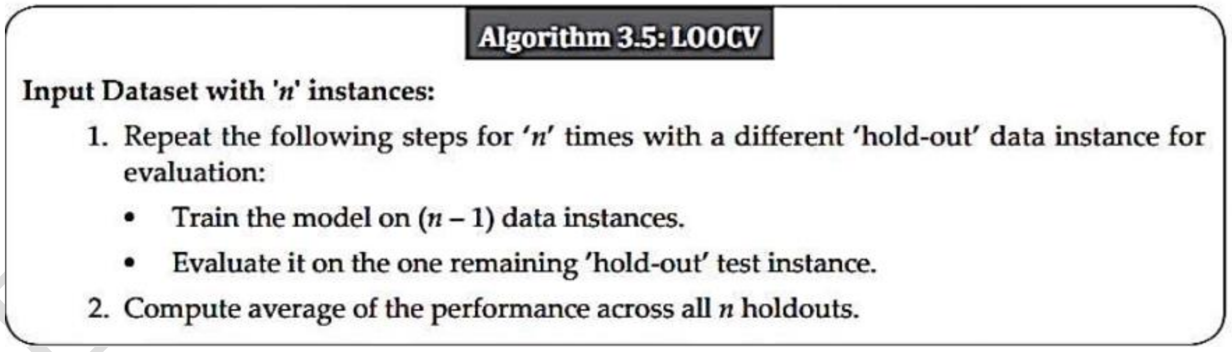


Figure 3.5: Illustration of Leave-One-Out Cross-Validation



Algorithm 3.5: LOOCV

Input Dataset with 'n' instances:

1. Repeat the following steps for 'n' times with a different 'hold-out' data instance for evaluation:
   - Train the model on (n – 1) data instances.
   - Evaluate it on the one remaining 'hold-out' test instance.
2. Compute average of the performance across all n holdouts.

Visualizing Model Performance

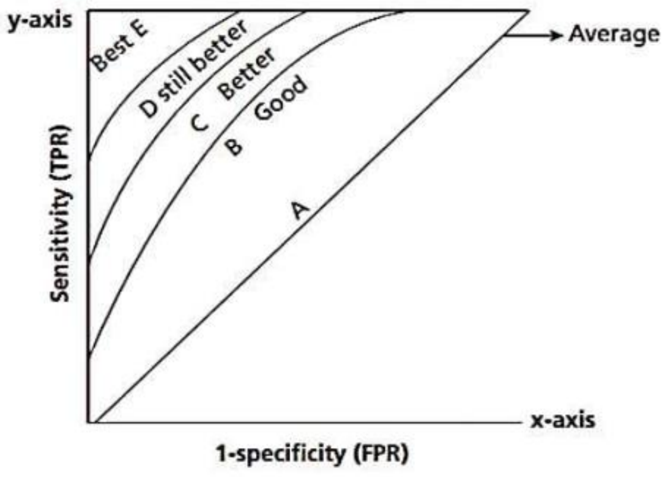


Figure 3.6: A Sample ROC Curve

ROC Curve (Receiver Operating Characteristic):

- Plots **True Positive Rate** vs. **False Positive Rate**.
- **Area Under the Curve (AUC)**: Measures classifier performance (1.0 = perfect, closer to diagonal = less accurate).

Precision-Recall Curve:

- Useful for imbalanced datasets to evaluate precision and recall.

Scoring and Complexity Methods

Scoring Models: Combine model performance and complexity into a single score.

Example: **Minimum Description Length (MDL)**:

Selects the simplest model with the fewest bits to represent both data and predictions.

Formula:

$$MDL = -\log(p(x)) - \log(p(y|x, \theta))$$

where x = input, y = target, and θ = model parameters.