Machine Learning Course
23CSM241

Prof Rajani Kanth VELURU

Senior Member (LM), Union of Radio Scientific International, Belgium

J. Sheik Mohamed

V. Raghubathy

Department of CSE-AIML
Sreenivasa Institute of Technology And Management Studies

March 9, 2026

Contents

1 Introduction to Machine Learning **8**

1.1 Evolution of Machine Learning . . . 8
1.1.1 Historical Timeline and Key Milestones . . . 8
1.1.2 Mathematical Foundations . . . 8
1.2 Paradigms for Machine Learning . . . 8
1.2.1 Formal Definition . . . 8
1.2.2 Major Learning Paradigms . . . 10
1.3 Learning by Rote: Mathematical Formulation and Analysis . . . 10
1.3.1 Formal Definition and Mathematical Characterization . . . 10
1.3.2 Mathematical Properties and Theoretical Analysis . . . 11
1.3.3 Example Problem 1: Rigorous Analysis . . . 12
1.3.4 Applications and Real-World Instantiations . . . 13
1.3.5 Limitations and Fundamental Constraints . . . 14
1.3.6 Hybrid Approaches and Extensions . . . 14
1.3.7 Conclusion and Philosophical Implications . . . 15
1.4 Learning by Induction: Mathematical Framework . . . 16
1.4.1 Foundations of Inductive Learning . . . 16
1.4.2 Statistical Learning Theory . . . 17
1.4.3 Bias-Variance Decomposition: Detailed Analysis . . . 18
1.4.4 Linear Regression: Comprehensive Mathematical Treatment . . . 19
1.4.5 Regularization and Model Complexity Control . . . 20
1.4.6 Generalization Beyond Linear Models . . . 21
1.4.7 Empirical Risk Minimization (ERM) and Alternatives . . . 21
1.4.8 Applications and Modern Context . . . 22
1.4.9 Conclusion and Fundamental Insights . . . 22
1.5 Comparative Analysis . . . 23
1.5.1 Mathematical Comparison . . . 23
1.5.2 Practical Example: Learning the Sine Function . . . 23
1.5.3 Theoretical Insights . . . 24
1.6 Advanced Topics . . . 24
1.6.1 Kernel Methods and Implicit Generalization . . . 24
1.6.2 Neural Networks as Inductive Learners . . . 24
1.6.3 Bayesian Learning . . . 24
1.7 Conclusion . . . 24

1.7.1 Key Takeaways . . . 24
1.7.2 Future Directions . . . 25
1.8 Reinforcement Learning: Mathematical Foundations . . . 25
1.8.1 Markov Decision Process (MDP) Formulation . . . 25
1.8.2 Bellman Optimality Equations . . . 26
1.9 Types of Data in Machine Learning . . . 26
1.9.1 Mathematical Characterization of Data Types . . . 26
1.9.2 Data Transformation Techniques . . . 33
1.10 Stages in Machine Learning Pipeline . . . 33
1.10.1 Formal Mathematical Framework . . . 33
1.10.2 Detailed Stage-by-Stage Analysis . . . 34
1.11 Mathematical Problems and Solutions . . . 35
1.11.1 Problem Set 1: Basic Calculations . . . 35
1.12 Multiple Choice Questions (MCQs) . . . 35
1.12.1 Knowledge Level Questions (Remembering) . . . 35
1.12.2 Comprehension Level Questions (Understanding) . . . 36
1.12.3 Application Level Questions (Applying) . . . 36
1.12.4 Analysis Level Questions (Analyzing) . . . 37
1.12.5 Synthesis Level Questions (Evaluating) . . . 37
1.13 Essay Questions and Solutions . . . 37
1.13.1 Long Answer Questions . . . 37
1.14 Applications and Case Studies . . . 39
1.14.1 Real-World Applications . . . 39
1.14.2 Case Study: Email Spam Detection . . . 40
1.15 Summary and Key Takeaways . . . 40
1.15.1 Key Mathematical Concepts . . . 40
1.15.2 Important Formulas . . . 40
1.15.3 Practical Guidelines . . . 41

2 Nearest Neighbour-Based Models **42**
2.1 Introduction to Proximity Measures . . . 42
2.1.1 Types of Proximity Measures . . . 42
2.2 Mathematical Foundations of Distance Measures . . . 42
2.2.1 Formal Definition of a Metric . . . 42
2.2.2 Common Distance Measures . . . 43
2.2.3 Specialized Distance Measures . . . 43
2.2.4 Theoretical Analysis of Distance Measures . . . 44
2.3 Similarity Measures and Correlation Functions . . . 44
2.3.1 Non-Metric Similarity Functions . . . 44
2.3.2 Proximity Between Binary Patterns . . . 45
2.3.3 Mathematical Properties of Similarity Measures . . . 46
2.4 Theoretical Foundations of Nearest Neighbor Methods . . . 46
2.4.1 Mathematical Framework . . . 46
2.4.2 Convergence Properties . . . 46
2.5 K-Nearest Neighbor Classifier: Mathematical Analysis . . . 47

2.5.1 Formal Definition . . . 47
2.5.2 Algorithm with Weighted Voting . . . 47
2.5.3 Bias-Variance Analysis for k-NN . . . 48
2.5.4 Computational Complexity Analysis . . . 48
2.6 Radius-Based Nearest Neighbor Algorithm . . . 49
2.6.1 Motivation and Mathematical Formulation . . . 49
2.6.2 Algorithm with Adaptive Radius . . . 49
2.6.3 Theoretical Analysis . . . 49
2.7 K-Nearest Neighbor Regression . . . 50
2.7.1 Mathematical Formulation . . . 50
2.7.2 Weighted k-NN Regression . . . 50
2.7.3 Convergence Analysis . . . 51
2.7.4 Example: Detailed Calculation . . . 51
2.8 Advanced Nearest Neighbor Variants . . . 51
2.8.1 Learning Vector Quantization (LVQ) . . . 51
2.8.2 Radial Basis Function Networks . . . 52
2.9 Theoretical Limitations and Challenges . . . 52
2.9.1 Curse of Dimensionality . . . 52
2.9.2 Sample Complexity Analysis . . . 53
2.10 Performance Evaluation of Nearest Neighbor Methods . . . 53
2.10.1 Error Decomposition . . . 53
2.10.2 Cross-Validation for Parameter Tuning . . . 53
2.10.3 Confidence Intervals for k-NN . . . 53
2.11 Applications and Case Studies . . . 54
2.11.1 Example 1: Image Classification with k-NN . . . 54
2.11.2 Example 2: Collaborative Filtering . . . 54
2.11.3 Example 3: Anomaly Detection . . . 54
2.12 Mathematical Problems and Solutions . . . 54
2.12.1 Problem Set 1: Distance Calculations . . . 54
2.12.2 Problem Set 2: k-NN Calculations . . . 55
2.13 Multiple Choice Questions . . . 55
2.13.1 Knowledge Level Questions . . . 55
2.13.2 Comprehension Level Questions . . . 56
2.13.3 Application Level Questions . . . 56
2.14 Essay Questions and Solutions . . . 56
2.15 Summary and Key Takeaways . . . 58
2.15.1 Mathematical Foundations . . . 58
2.15.2 Algorithmic Properties . . . 58
2.15.3 Theoretical Insights . . . 58
2.15.4 Practical Guidelines . . . 58
2.15.5 Important Formulas . . . 59
2.15.6 Future Directions . . . 59

3 UNIT 3: MODELS BASED ON DECISION TREES **60**
3.1 INTRODUCTION TO DECISION TREES . . . 61
3.1.1 Basic Structure . . . 61
3.1.2 Decision Tree Learning Algorithm . . . 61
3.2 IMPURITY MEASURES . . . 62
3.2.1 Entropy . . . 62
3.2.2 Gini Index . . . 63
3.2.3 Misclassification Error . . . 63
3.2.4 Comparison of Impurity Measures . . . 64
3.3 INFORMATION GAIN AND SPLITTING CRITERIA . . . 64
3.3.1 Information Gain . . . 64
3.3.2 Gain Ratio . . . 65
3.3.3 Worked Example: Building a Decision Tree . . . 65
3.4 PROPERTIES OF DECISION TREES . . . 70
3.4.1 Advantages . . . 70
3.4.2 Disadvantages . . . 70
3.5 REGRESSION BASED ON DECISION TREES . . . 71
3.5.1 Regression Tree Structure . . . 71
3.5.2 Example of Regression Tree . . . 71
3.6 BIAS-VARIANCE TRADEOFF . . . 72
3.6.1 Definitions . . . 72
3.6.2 The Tradeoff . . . 72
3.6.3 Decision Tree Complexity . . . 72
3.7 RANDOM FORESTS FOR CLASSIFICATION AND REGRESSION . . . 73
3.7.1 Bootstrap Aggregation (Bagging) . . . 73
3.7.2 Bagging (Bootstrap Aggregating) . . . 73
3.7.3 Random Forest Algorithm . . . 75
3.7.4 Out-of-Bag Error . . . 76
3.7.5 Feature Importance . . . 76
3.7.6 Convergence Property . . . 76
3.8 THE BAYES CLASSIFIER . . . 76
3.8.1 Introduction to Bayes Classifier . . . 76
3.8.2 Bayes' Rule and Inference . . . 77
3.8.3 Bayes Classifier Optimality . . . 77
3.8.4 Bayes Decision Boundary . . . 77
3.8.5 Multi-Class Classification . . . 78
3.9 NAIVE BAYES CLASSIFIER . . . 78
3.9.1 Class Conditional Independence Assumption . . . 78
3.9.2 Naive Bayes Classifier (NBC) . . . 78
3.9.3 Estimating Probabilities . . . 78
3.9.4 Worked Example: Spam Classification . . . 79
3.9.5 Gaussian Naive Bayes Example . . . 80
3.9.6 Advantages and Disadvantages of Naive Bayes . . . 82
3.10 CASE STUDIES . . . 82
3.10.1 Case Study 1: Credit Risk Assessment . . . 82

3.10.2 Case Study 2: Medical Diagnosis . . . 83
3.10.3 Case Study 3: Customer Churn Prediction . . . 83
3.11 EXERCISES . . . 83
3.11.1 Theoretical Exercises . . . 83
3.11.2 Practical Exercises . . . 84
3.12 SOLUTIONS TO SELECTED EXERCISES . . . 84
3.12.1 Solution to Exercise 1.1 . . . 84
3.12.2 Solution to Exercise 2.2 . . . 84

4 Linear Discriminants for Machine Learning 86
4.1 Introduction to Linear Discriminants . . . 86
4.1.1 Mathematical Framework . . . 86
4.1.2 Geometric Interpretation . . . 86
4.2 Linear Discriminants for Classification . . . 87
4.2.1 Fisher's Linear Discriminant . . . 87
4.2.2 Mathematical Derivation . . . 87
4.2.3 Example Problem 4.1: Fisher's Linear Discriminant . . . 87
4.3 Perceptron Classifier . . . 88
4.3.1 The Perceptron Model . . . 88
4.3.2 Geometric Interpretation . . . 89
4.4 Perceptron Learning Algorithm . . . 89
4.4.1 Algorithm Description . . . 89
4.4.2 Convergence Theorem . . . 89
4.4.3 Example Problem 4.2: Perceptron Learning . . . 90
4.5 Support Vector Machines (SVM) . . . 90
4.5.1 Mathematical Formulation . . . 90
4.5.2 Geometric Interpretation . . . 91
4.5.3 Optimization Problem . . . 91
4.5.4 Dual Formulation . . . 91
4.5.5 Example Problem 4.3: SVM Computation . . . 91
4.6 Linearly Non-Separable Case . . . 92
4.6.1 Soft Margin SVM . . . 92
4.6.2 Dual Formulation with Slack . . . 92
4.6.3 Interpretation of C . . . 92
4.7 Non-linear SVM and Kernel Trick . . . 92
4.7.1 Motivation . . . 92
4.7.2 Kernel Trick . . . 92
4.7.3 Common Kernel Functions . . . 93
4.7.4 Mercer's Theorem . . . 93
4.7.5 Example Problem 4.4: Kernel SVM . . . 93
4.8 Logistic Regression . . . 93
4.8.1 Mathematical Model . . . 93
4.8.2 Likelihood and Loss Function . . . 93
4.8.3 Gradient Derivation . . . 94
4.8.4 Example Problem 4.5: Logistic Regression . . . 94

4.9 Linear Regression (Review) . . . 94
4.9.1 Matrix Formulation . . . 94
4.9.2 Ordinary Least Squares Solution . . . 94
4.9.3 Connection to Logistic Regression . . . 95
4.10 Multi-Layer Perceptrons (MLPs) . . . 95
4.10.1 Architecture . . . 95
4.10.2 Activation Functions . . . 95
4.10.3 Universal Approximation Theorem . . . 95
4.11 Backpropagation for Training an MLP . . . 96
4.11.1 Forward Propagation . . . 96
4.11.2 Loss Function . . . 96
4.11.3 Backpropagation Algorithm . . . 96
4.11.4 Example Problem 4.6: Backpropagation . . . 97
4.11.5 Optimization Algorithms . . . 98
4.12 Regularization in Neural Networks . . . 99
4.12.1 L1 and L2 Regularization . . . 99
4.12.2 Dropout . . . 99
4.12.3 Batch Normalization . . . 99
4.13 Early Stopping . . . 100
4.14 Multiple Choice Questions . . . 100
4.14.1 Knowledge Level Questions . . . 100
4.14.2 Comprehension Level Questions . . . 100
4.14.3 Application Level Questions . . . 101
4.14.4 Analysis Level Questions . . . 101
4.14.5 Synthesis Level Questions . . . 101
4.15 Essay Questions and Solutions . . . 102
4.16 Summary and Key Takeaways . . . 103
4.16.1 Mathematical Foundations . . . 103
4.16.2 Key Insights . . . 104
4.16.3 Important Formulas . . . 104
4.16.4 Practical Guidelines . . . 104

5 Clustering **105**
5.1 Introduction to Clustering . . . 105
5.1.1 Mathematical Framework . . . 105
5.1.2 Objective Functions . . . 105
5.2 Partitioning of Data . . . 105
5.2.1 Data Representation . . . 105
5.3 Matrix Factorization for Clustering . . . 106
5.3.1 Spectral Decomposition . . . 106
5.3.2 Non-negative Matrix Factorization (NMF) . . . 106
5.3.3 Example Problem 5.1: Matrix Factorization . . . 106
5.4 Clustering of Patterns . . . 107
5.4.1 Feature Space . . . 107
5.4.2 Similarity Measures . . . 107

5.5 Hierarchical Clustering . . . 107
5.5.1 Distance Between Clusters . . . 107
5.5.2 Divisive Clustering (Top-Down) . . . 109
5.5.3 Agglomerative Clustering (Bottom-Up) . . . 109
5.5.4 Example Problem 5.2: Agglomerative Clustering . . . 109
5.6 Partitional Clustering . . . 109
5.6.1 K-Means Clustering . . . 109
5.6.2 Convergence Properties . . . 110
5.6.3 Example Problem 5.3: K-Means Clustering . . . 110
5.7 Soft Partitioning and Soft Clustering . . . 111
5.7.1 Fuzzy C-Means Clustering . . . 111
5.7.2 Example Problem 5.4: Fuzzy C-Means . . . 112
5.7.3 Rough Clustering . . . 112
5.7.4 Rough K-Means Clustering Algorithm . . . 113
5.8 Expectation Maximization-Based Clustering . . . 113
5.8.1 Gaussian Mixture Models (GMM) . . . 113
5.8.2 EM Algorithm for GMM . . . 114
5.8.3 Example Problem 5.5: EM Algorithm . . . 114
5.9 Spectral Clustering . . . 115
5.9.1 Graph-Based Formulation . . . 115
5.9.2 Similarity Matrix . . . 115
5.9.3 Graph Laplacian . . . 115
5.9.4 Spectral Clustering Algorithm . . . 116
5.9.5 Example Problem 5.6: Spectral Clustering . . . 116
5.10 Comparison of Clustering Methods . . . 117
5.11 Evaluation of Clustering . . . 117
5.11.1 Internal Validation Measures . . . 117
5.11.2 External Validation Measures . . . 118
5.12 Multiple Choice Questions . . . 118
5.12.1 Knowledge Level Questions . . . 118
5.12.2 Comprehension Level Questions . . . 118
5.12.3 Application Level Questions . . . 119
5.12.4 Analysis Level Questions . . . 119
5.13 Essay Questions and Solutions . . . 120
5.14 Summary and Key Takeaways . . . 122
5.14.1 Mathematical Foundations . . . 122
5.14.2 Important Formulas . . . 123
5.14.3 Practical Guidelines . . . 123

Chapter 1

Introduction to Machine Learning

1.1 Evolution of Machine Learning

1.1.1 Historical Timeline and Key Milestones

Machine Learning (ML) has evolved through several distinct phases, each characterized by breakthroughs in theory, algorithms, and computing capability.

1.1.2 Mathematical Foundations

The evolution of ML is deeply rooted in mathematical advances:

- **Linear Algebra**: Matrix operations, eigenvalues, SVD for dimensionality reduction
- **Probability Theory**: Bayesian inference, Markov models
- **Statistics**: Regression analysis, hypothesis testing
- **Calculus**: Gradient descent, optimization theory
- **Information Theory**: Entropy, mutual information

1.2 Paradigms for Machine Learning

1.2.1 Formal Definition

Machine Learning is formally defined as:

Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Mathematically, this can be expressed as:

$$\exists f : X \to Y \quad \text{such that} \quad P(f(x) \approx y) \geq 1 - \epsilon$$

where f is learned from training data $D = \{(x_i, y_i)\}_{i=1}^{n}$.

Table 1.1: Evolution of Machine Learning: Key Milestones

Era	**Key Developments**	**Pioneers/Contributors**
1940s-1950s	• McCulloch-Pitts neuron model (1943) • Hebbian learning rule (1949) • First neural network (SNARC, 1951)	Warren McCulloch, Walter Pitts, Donald Hebb
1950s-1960s	• Turing's learning machine concept (1950) • Perceptron invented (1957) • First ML program (checkers, 1959) • "Machine Learning" term coined (1959)	Alan Turing, Frank Rosenblatt, Arthur Samuel
1970s-1980s	• Backpropagation algorithm (1974) • Decision tree algorithms (ID3, 1986) • PAC learning framework (1984) • Connectionism revival	Paul Werbos, J. Ross Quinlan, Leslie Valiant, Rumelhart & McClelland
1990s-2000s	• Support Vector Machines (1992) • Random Forests (1995) • Boosting algorithms (1997) • Graphical models • Kernel methods	Vapnik & Cortes, Leo Breiman, Yoav Freund & Schapire, Pearl, Jordan
2010s-Present	• Deep Learning resurgence • Big Data analytics • Transfer learning • Explainable AI • Generative models (GANs, Transformers)	Hinton, Bengio, LeCun, Goodfellow, Vaswani

1.2.2 Major Learning Paradigms

1. **Symbolic Learning**: Rule-based systems, inductive logic programming

   Example: Learning decision rules:

   $$\text{IF } (x_1 > \theta_1) \wedge (x_2 < \theta_2) \text{ THEN class} = \text{A}$$

2. **Statistical Learning**: Probabilistic models, Bayesian networks

   Example: Naive Bayes classifier:

   $$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \propto P(y) \prod_{j=1}^{d} P(x_j|y)$$

3. **Connectionist Learning**: Neural networks, deep learning

   Example: Simple neuron model:

   $$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

   where σ is an activation function.

4. **Evolutionary Learning**: Genetic algorithms, swarm intelligence

   Example: Fitness function evaluation:

   $$f(\mathbf{x}) = \sum_{i=1}^{n} w_i g_i(\mathbf{x})$$

1.3 Learning by Rote: Mathematical Formulation and Analysis

1.3.1 Formal Definition and Mathematical Characterization

Rote learning represents a non-parametric learning paradigm characterized by exact memorization without abstraction or generalization mechanisms. Given a training dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ sampled from an unknown distribution $\mathcal{P}(X, Y)$, the learned function $f_{\text{rote}} : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$ is formally defined as:

$$f_{\text{rote}}(x) = \begin{cases} y_i & \text{if } \exists (x_i, y_i) \in D \text{ such that } \delta(x, x_i) = 1 \\ \perp & \text{otherwise} \end{cases}$$

where:

- $\mathcal{X} \subseteq \mathbb{R}^d$ denotes the d-dimensional input feature space
- $\mathcal{Y}$ denotes the output space (for regression: $\mathcal{Y} \subseteq \mathbb{R}$; for classification: $\mathcal{Y} = \{1, 2, \ldots, K\}$)

- $\perp$ represents "undefined" or "no output" (formalizing the absence of generalization)
- $\delta : \mathcal{X} \times \mathcal{X} \to \{0, 1\}$ is an exact identity metric:

$$\delta(x, x') = \mathbb{I}_{\{x=x'\}} = \begin{cases} 1 & \text{if } \|x - x'\|_0 = 0 \quad \text{(exact component-wise equality)} \\ 0 & \text{otherwise} \end{cases}$$

where $\|x - x'\|_0$ counts the number of differing components (Hamming distance for discrete features, exact equality for continuous)

1.3.2 Mathematical Properties and Theoretical Analysis

Hypothesis Space and Capacity

The hypothesis space of rote learning is:

$$\mathcal{H}_{\text{rote}} = \{h : \mathcal{X} \to \mathcal{Y} \cup \{\perp\} \mid h(x) = y_i \text{ for } x = x_i \in D, \text{ and undefined otherwise}\}$$

VC Dimension Analysis:

- For continuous input spaces $\mathcal{X} \subseteq \mathbb{R}^d$: VC-dim$(\mathcal{H}_{\text{rote}}) = \infty$

  Sketch: For any set of m distinct points $\{x_1, \ldots, x_m\} \subset \mathcal{X}$, assign arbitrary labels $y_1, \ldots, y_m \in \mathcal{Y}$. The rote learner can memorize this exact mapping. Since m is arbitrary, the VC dimension is unbounded. □

- For discrete input spaces with $|\mathcal{X}| = M$: VC-dim$(\mathcal{H}_{\text{rote}}) = M$
- For Boolean input spaces $\mathcal{X} = \{0, 1\}^d$: VC-dim$(\mathcal{H}_{\text{rote}}) = 2^d$

Empirical Risk and Generalization Bounds

- **Empirical Risk (Training Error)**:

$$R_{\text{emp}}(f_{\text{rote}}) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_{\text{rote}}(x_i)) = 0$$

for any loss function L satisfying $L(y, y) = 0$ (e.g., 0-1 loss, squared error).

- **Generalization Error**:

$$R(f_{\text{rote}}) = \mathbb{E}_{(x,y)\sim\mathcal{P}}[L(y, f_{\text{rote}}(x))]$$

Since f_{rote} is undefined for $x \notin \{x_1, \ldots, x_n\}$, we define:

$$R(f_{\text{rote}}) = \mathbb{P}(x \notin \{x_1, \ldots, x_n\}) \cdot \mathbb{E}[L(y, \text{default}) \mid x \text{ unseen}]$$

where "default" represents a baseline prediction (e.g., mean of training outputs for regression, majority class for classification).

- **Generalization Gap**:

$$R(f_{\text{rote}}) - R_{\text{emp}}(f_{\text{rote}}) = R(f_{\text{rote}}) \geq 0$$

The gap equals the true risk, highlighting the absence of generalization.

Statistical Learning Theory Perspective

From the Probably Approximately Correct (PAC) learning framework:

- **Sample Complexity**: For any $\epsilon > 0$, $\delta > 0$, to achieve $R(f) \leq \epsilon$ with probability at least $1 - \delta$, rote learning requires:

$$n \geq \frac{1}{\epsilon} \ln\left(\frac{1}{\delta}\right)$$

  assuming the test distribution matches the empirical distribution of training points.

- **No Free Lunch Theorem**: Averaged over all possible data-generating distributions, rote learning has the same expected error as any other learning algorithm when evaluated on unseen data.

1.3.3 Example Problem 1: Rigorous Analysis

Problem Statement

Given training set: $D = \{(1, 2), (2, 4), (3, 6)\} \subset \mathbb{R} \times \mathbb{R}$

Define the rote learning function $f_{\text{rote}} : \mathbb{R} \to \mathbb{R} \cup \{\perp\}$ and evaluate $f_{\text{rote}}(2.5)$.

Formal Solution

$$\begin{aligned}
\text{Training set: } & D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3)\} = \{(1, 2), (2, 4), (3, 6)\} \\
\text{Input: } & x = 2.5 \\
\text{Exact matching: } & \delta(2.5, x_i) = 0 \quad \forall i \in \{1, 2, 3\} \\
\text{Therefore: } & f_{\text{rote}}(2.5) = \perp \quad \text{(undefined)}
\end{aligned}$$

Computational Complexity Analysis

- **Training time**: $O(n)$ to store n examples
- **Inference time**:
  - Naive lookup: $O(n)$ for linear search
  - With hash table: $O(1)$ average case (exact match)
  - With balanced tree: $O(\log n)$ for ordered retrieval
- **Space complexity**: $O(n)$ to store all examples

1.3.4 Applications and Real-World Instantiations

Exact Pattern Matching Systems

1. **Associative Memory / Content-Addressable Memory**:
   - Mathematical model: $M : \mathcal{X} \to \mathcal{Y}$ with $M(x) = y$ if $(x, y) \in D$
   - Implementations: Hash tables, Bloom filters (probabilistic variant)
   - Applications: Database indexing, DNS resolution
2. **Memoization in Dynamic Programming**:
   - Function $F : \mathcal{X} \to \mathcal{Y}$ with cache $C \subseteq \mathcal{X} \times \mathcal{Y}$
   - Algorithm:
     Pseudocode:

```
if x in domain(C):
return C(x)
else:
y = OriginalComputation(x)
C(x) = y
return y
```

3. **Perfect Hashing Systems**:
   - Construct hash function $h : \mathcal{X} \to \{1, \ldots, m\}$ with no collisions for known set $S \subset \mathcal{X}$
   - Guarantees $O(1)$ retrieval for exact matches
4. **Deterministic Finite Automata (DFA) for String Matching**:
   - Accepts exactly the strings in training set
   - Example: Aho-Corasick algorithm for multiple exact string matching

When Rote Learning is Optimal

- **Discrete Uniform Distribution**: When $\mathcal{P}(x)$ is uniform over finite set S and all $x \in S$ appear in training data
- **Deterministic Functions**: When $y = f(x)$ is deterministic and all possible x are observed
- **Noise-Free Environments**: When measurement or labeling noise is absent
- **Small Domain Size**: When $|\mathcal{X}|$ is small enough to enumerate

1.3.5 Limitations and Fundamental Constraints

Theoretical Limitations

1. **Absence of Generalization**:

   $$\forall x \notin \{x_1, \ldots, x_n\}, \quad f_{\text{rote}}(x) = \perp$$

   This violates the fundamental goal of machine learning: to perform well on unseen data.

2. **Curse of Dimensionality**: For $\mathcal{X} = [0,1]^d$, to cover the space with density ρ (points per unit volume), we need:

   $$n = \rho^d \quad \Rightarrow \quad \text{Exponential growth with dimension}$$

3. **No Smoothness Prior**: Rote learning assumes no relationship between nearby points:

   $$\lim_{x' \to x} f_{\text{rote}}(x') \neq f_{\text{rote}}(x) \quad \text{in general}$$

4. **Poor Sample Efficiency**: The estimation error (difference between empirical and expected risk) doesn't decrease with n for unseen regions.

Practical Limitations

- **Storage Inefficiency**: Requires $O(n)$ memory, where n is dataset size
- **Sensitivity to Noise**: For noisy observations $y_i = f(x_i) + \epsilon_i$, rote learning memorizes noise:

  $$\mathbb{E}[(f_{\text{rote}}(x_i) - f(x_i))^2] = \text{Var}(\epsilon)$$

- **Poor Scalability**: Inference time grows with n unless using advanced data structures
- **Inability to Handle Approximate Inputs**: Slight variations in input (e.g., 2.5000001 vs 2.5) yield no output

Formal Comparison with Inductive Learning

Let $\mathcal{H}_{\text{ind}}$ be a parametric hypothesis class (e.g., linear functions). Then:

1.3.6 Hybrid Approaches and Extensions

k-Nearest Neighbors (k-NN) as a Smoothing Extension

k-NN can be viewed as a generalization of rote learning with $k = 1$:

$$f_{k\text{-NN}}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

where $N_k(x)$ are the k nearest neighbors of x.

Properties:

Property	Rote Learning ($\mathcal{H}_{\text{rote}}$)	Inductive Learning ($\mathcal{H}_{\text{ind}}$)
Hypothesis Space	Non-parametric, data-dependent	Parametric, fixed before seeing data
VC Dimension	∞ (continuous), $\lvert\mathcal{X}\rvert$ (discrete)	Finite (e.g., $d+1$ for linear regression in $\mathbb{R}^d$)
Empirical Risk	0 (memorizes training data)	≥ 0 (approximates training data)
Generalization	None (undefined for new points)	Yes (predicts for all $x \in \mathcal{X}$)
Sample Complexity	$O(1/\epsilon)$ for exact match	$O(\frac{d}{\epsilon}\log\frac{1}{\delta})$ for PAC learning
Smoothness	No continuity assumptions	Often assumes f is smooth/Lipschitz

Table 1.2: Formal comparison between rote and inductive learning paradigms

- For $k = 1$: Reduces to rote learning with distance-based matching
- For $k > 1$: Introduces local averaging (smoothing)
- As $k \to \infty$: Approaches global average (maximum smoothing)

Kernel Methods and Similarity-Based Learning

Generalization using kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$f_{\text{kernel}}(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$$

where rote learning corresponds to $K(x, x_i) = \delta(x, x_i)$.

Overfitting as Rote Learning in Disguise

A complex model (e.g., deep neural network with many parameters) that achieves zero training error is effectively performing rote learning:

$$\lim_{\text{capacity}\to\infty} f_{\text{complex}}(x) \approx f_{\text{rote}}(x) \quad \text{on training points}$$

1.3.7 Conclusion and Philosophical Implications

The Role of Rote Learning in Cognitive Science

- **Human Memory**: Episodic memory exhibits rote-like properties for specific experiences
- **Skill Acquisition**: Early stages often involve rote memorization before pattern recognition emerges
- **Education Theory**: Rote learning vs. conceptual understanding debate

Fundamental Tradeoffs in Learning Systems

The rote-inductive spectrum illustrates fundamental tradeoffs:

$$\begin{aligned} \text{Memorization} &\leftrightarrow \text{Generalization} \\ \text{Flexibility} &\leftrightarrow \text{Stability} \\ \text{Low bias} &\leftrightarrow \text{Low variance} \\ \text{Specificity} &\leftrightarrow \text{Transferability} \end{aligned}$$

Mathematical Epilogue

Rote learning represents the extreme case in the bias-variance decomposition:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x)) + \sigma^2$$

For rote learning: $\text{Bias}^2 \approx 0$ on training points, Var is high (sensitive to specific examples), and generalization requires seeing exact inputs.

Note: While rote learning appears simplistic, its study provides fundamental insights into the nature of learning, memorization, and the essential role of inductive bias in generalization. It serves as a critical baseline against which more sophisticated learning algorithms can be evaluated.

1.4 Learning by Induction: Mathematical Framework

1.4.1 Foundations of Inductive Learning

The Inductive Learning Hypothesis

The fundamental principle underlying inductive learning, formalized as the *Inductive Learning Hypothesis*, states:

If a hypothesis $h \in \mathcal{H}$ approximates the target function f sufficiently wellon a sufficiently large training s

Formally, for $\epsilon > 0$, $\delta > 0$, we require:

$$\mathbb{P}\left(R(h) - R_{\text{emp}}(h) > \epsilon\right) \leq \delta$$

where $R(h) = \mathbb{E}_{(x,y)\sim P}[\ell(h(x), y)]$ is the expected risk (generalization error) and $R_{\text{emp}}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i), y_i)$ is the empirical risk.

Formal Learning Problem Statement

Given:

- **Input space**: $\mathcal{X} \subseteq \mathbb{R}^d$ (d-dimensional feature space)
- **Output space**: $\mathcal{Y}$ (continuous for regression, discrete for classification)
- **Hypothesis class**: $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y} \mid h \text{ parameterized by } \theta \in \Theta\}$
- **Loss function**: $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ measuring prediction error
- **Training data**: $D = \{(x_i, y_i)\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} P_{\mathcal{X}\times\mathcal{Y}}$
- **Underlying distribution**: P over $\mathcal{X} \times \mathcal{Y}$ (unknown)

The learning objective is to find:

$$h^* = \arg\min_{h\in\mathcal{H}} R(h) = \arg\min_{h\in\mathcal{H}} \mathbb{E}_{(x,y)\sim P}[\ell(h(x), y)]$$

Since P is unknown, we minimize the empirical risk:

$$h_n^* = \arg\min_{h\in\mathcal{H}} R_{\text{emp}}(h) = \arg\min_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i), y_i)$$

1.4.2 Statistical Learning Theory

Vapnik-Chervonenkis (VC) Theory

The VC dimension $d_{\text{VC}}(\mathcal{H})$ measures the capacity of a hypothesis class:

Definition 1.4.1 (VC Dimension)**.** *The VC dimension of $\mathcal{H}$ is the maximum number of points that can be shattered by $\mathcal{H}$:*

$$d_{VC}(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\}$$

where $\Pi_{\mathcal{H}}(m)$ is the growth function counting the maximum number of dichotomies on m points.

Generalization Error Bounds

For binary classification with 0-1 loss, with probability at least $1 - \delta$:

$$R(h) \leq R_{\text{emp}}(h) + \sqrt{\frac{d_{\text{VC}}(\mathcal{H})\left(\log\left(\frac{2n}{d_{\text{VC}}(\mathcal{H})}\right) + 1\right) - \log\left(\frac{\delta}{4}\right)}{n}}$$

More generally, for bounded loss $\ell \in [0, M]$:

$$R(h) \leq R_{\text{emp}}(h) + \mathcal{R}_n(\mathcal{H}) + M\sqrt{\frac{\log(1/\delta)}{2n}}$$

where $\mathcal{R}_n(\mathcal{H}) = \mathbb{E}_{\sigma,D}[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^n \sigma_i \ell(h(x_i), y_i)]$ is the Rademacher complexity.

PAC Learning Framework

A hypothesis class $\mathcal{H}$ is *Probably Approximately Correct (PAC) learnable* if there exists an algorithm $\mathcal{A}$ such that for any $\epsilon > 0$, $\delta > 0$, and any distribution P, when given $n \geq \text{poly}(1/\epsilon, 1/\delta, \text{size}(c), d_{\text{VC}}(\mathcal{H}))$ samples, $\mathcal{A}$ outputs h with:

$$\mathbb{P}\left(R(h) \leq \min_{h' \in \mathcal{H}} R(h') + \epsilon\right) \geq 1 - \delta$$

1.4.3 Bias-Variance Decomposition: Detailed Analysis

Derivation and Interpretation

For squared error loss, consider the expected prediction error at a fixed point x:

$$\begin{aligned}
\mathbb{E}_D[(h_D(x) - f(x))^2] &= \mathbb{E}_D[(h_D(x) - \mathbb{E}_D[h_D(x)] + \mathbb{E}_D[h_D(x)] - f(x))^2] \\
&= \mathbb{E}_D[(h_D(x) - \mathbb{E}_D[h_D(x)])^2] + (\mathbb{E}_D[h_D(x)] - f(x))^2 \\
&\quad + 2\mathbb{E}_D[(h_D(x) - \mathbb{E}_D[h_D(x)])(\mathbb{E}_D[h_D(x)] - f(x))] \\
&= \underbrace{\mathbb{E}_D[(h_D(x) - \mathbb{E}_D[h_D(x)])^2]}_{\text{Variance}(x)} + \underbrace{(\mathbb{E}_D[h_D(x)] - f(x))^2}_{\text{Bias}^2(x)}
\end{aligned}$$

Adding the irreducible error $\sigma^2 = \mathbb{E}[(y - f(x))^2]$ gives the complete decomposition.

Geometric Interpretation in Hypothesis Space

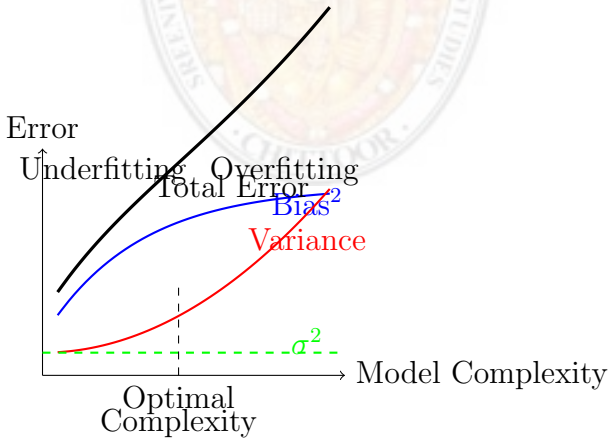


Figure 1.1: Bias-Variance tradeoff illustrating the U-shaped curve of total error

Implications for Model Selection

- **Simple models** (low complexity): High bias, low variance
- **Complex models** (high complexity): Low bias, high variance
- **Optimal model**: Minimizes sum of bias^2, variance, and irreducible error

1.4.4 Linear Regression: Comprehensive Mathematical Treatment

Problem Statement and Matrix Formulation

Given training data $D = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.

Assume linear model: $h(x) = w^T x + b$, where $w \in \mathbb{R}^d$, $b \in \mathbb{R}$.

In matrix notation with augmented feature vector $\tilde{x}_i = [1, x_i^T]^T$:

$$\tilde{X} = \begin{bmatrix} 1 & x_1^T \\ \vdots & \vdots \\ 1 & x_n^T \end{bmatrix} \in \mathbb{R}^{n \times (d+1)}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \theta = \begin{bmatrix} b \\ w \end{bmatrix}$$

The model becomes: $h(\tilde{X}) = \tilde{X}\theta$.

Ordinary Least Squares (OLS) Solution

Minimize the sum of squared residuals:

$$J(\theta) = \|y - \tilde{X}\theta\|_2^2 = (y - \tilde{X}\theta)^T (y - \tilde{X}\theta)$$

Taking gradient and setting to zero:

$$\nabla_\theta J(\theta) = -2\tilde{X}^T (y - \tilde{X}\theta) = 0$$

Normal equations:

$$\tilde{X}^T \tilde{X}\theta = \tilde{X}^T y$$

If $\tilde{X}^T \tilde{X}$ is invertible (full rank):

$$\hat{\theta} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

Example: Detailed Derivation

Given: $D = \{(1, 3), (2, 5), (3, 7), (4, 9)\}$

Step 1: Matrix formulation

$$\tilde{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}, \quad y = \begin{bmatrix} 3 \\ 5 \\ 7 \\ 9 \end{bmatrix}, \quad \theta = \begin{bmatrix} b \\ m \end{bmatrix}$$

Step 2: Compute $\tilde{X}^T \tilde{X}$ and $\tilde{X}^T y$

$$\tilde{X}^T \tilde{X} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$$

$$\tilde{X}^T y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 3 \\ 5 \\ 7 \\ 9 \end{bmatrix} = \begin{bmatrix} 24 \\ 70 \end{bmatrix}$$

Step 3: Solve normal equations

$$\begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix} = \begin{bmatrix} 24 \\ 70 \end{bmatrix}$$

Using Cramer's rule or matrix inversion:

$$\det(\tilde{X}^T \tilde{X}) = 4 \times 30 - 10 \times 10 = 120 - 100 = 20$$

$$m = \frac{\begin{vmatrix} 4 & 24 \\ 10 & 70 \end{vmatrix}}{20} = \frac{4 \times 70 - 10 \times 24}{20} = \frac{280 - 240}{20} = 2$$

$$b = \frac{\begin{vmatrix} 24 & 10 \\ 70 & 30 \end{vmatrix}}{20} = \frac{24 \times 30 - 10 \times 70}{20} = \frac{720 - 700}{20} = 1$$

Step 4: Final model and properties

$$h(x) = 2x + 1$$

Geometric interpretation: The solution $\hat{\theta}$ projects y onto the column space of $\tilde{X}$:

$$\hat{y} = \tilde{X}\hat{\theta} = \tilde{X}(\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y = Py$$

where P is the projection matrix.

Statistical Properties of OLS Estimator

- **Unbiasedness**: If $y = X\theta^* + \epsilon$ with $\mathbb{E}[\epsilon] = 0$, then $\mathbb{E}[\hat{\theta}] = \theta^*$
- **Variance**: $\text{Var}(\hat{\theta}) = \sigma^2 (X^T X)^{-1}$ under homoscedasticity
- **Gauss-Markov Theorem**: OLS is BLUE (Best Linear Unbiased Estimator)
- **Normality**: If $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, then $\hat{\theta} \sim \mathcal{N}(\theta^*, \sigma^2 (X^T X)^{-1})$

1.4.5 Regularization and Model Complexity Control

Tikhonov Regularization (Ridge Regression)

Add L2 penalty to control model complexity:

$$\hat{\theta}_{\text{ridge}} = \arg\min_{\theta} \left\{ \|y - X\theta\|_2^2 + \lambda \|\theta\|_2^2 \right\}$$

Closed-form solution:

$$\hat{\theta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y$$

LASSO (L1 Regularization)

Add L1 penalty for sparse solutions:

$$\hat{\theta}_{\text{lasso}} = \arg\min_{\theta} \left\{ \|y - X\theta\|_2^2 + \lambda \|\theta\|_1 \right\}$$

No closed-form solution in general; solved via coordinate descent or LARS.

Elastic Net

Combines L1 and L2 penalties:

$$\hat{\theta}_{\text{elastic}} = \arg\min_{\theta} \left\{ \|y - X\theta\|_2^2 + \lambda_1 \|\theta\|_1 + \lambda_2 \|\theta\|_2^2 \right\}$$

Bayesian Interpretation

Regularization corresponds to imposing priors on parameters:

- Ridge: Gaussian prior $\theta \sim \mathcal{N}(0, \tau^2 I)$
- LASSO: Laplace prior $p(\theta_i) \propto \exp(-\lambda|\theta_i|)$

1.4.6 Generalization Beyond Linear Models

The Kernel Trick

For non-linear relationships, map features to higher-dimensional space:

$$\phi : \mathcal{X} \to \mathcal{F}, \quad \dim(\mathcal{F}) \gg \dim(\mathcal{X})$$

Kernel function $K(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}$ allows computation without explicit ϕ.
Representer Theorem: Solution has form $h(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$.

Neural Networks as Universal Approximators

Feedforward neural networks with single hidden layer can approximate any continuous function on compact sets:

$$h(x) = \sum_{j=1}^{m} w_j \sigma \left(\sum_{i=1}^{d} v_{ij} x_i + b_j \right)$$

where σ is a non-polynomial activation function (sigmoid, ReLU, etc.).

1.4.7 Empirical Risk Minimization (ERM) and Alternatives

Structural Risk Minimization (SRM)

Minimize combination of empirical risk and complexity penalty:

$$h^* = \arg\min_{h \in \mathcal{H}} \{ R_{\text{emp}}(h) + \lambda C(h) \}$$

where $C(h)$ measures hypothesis complexity (e.g., norm in RKHS).

Regularized Risk Minimization

General formulation:

$$h^* = \arg\min_{h \in \mathcal{H}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i) + \lambda \Omega(h) \right\}$$

where $\Omega(h)$ is regularization term enforcing desired properties.

1.4.8 Applications and Modern Context

Deep Learning

Modern deep networks with millions of parameters challenge traditional generalization bounds:

- **Double descent phenomenon**: Risk decreases, increases, then decreases again with model complexity
- **Implicit regularization**: Optimization algorithms (SGD) induce regularization
- **Overparameterization**: Models with $p \gg n$ can still generalize well

Transfer Learning and Domain Adaptation

Leverage knowledge from source domain to improve learning in target domain:

$$h^* = \arg\min_{h} \{ R_{\text{source}}(h) + \lambda d(\mathcal{P}_{\text{source}}, \mathcal{P}_{\text{target}}) \}$$

Meta-Learning

Learn the learning algorithm itself:

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \mathbb{E}_{\text{task} \sim \mathcal{T}} [R(\mathcal{A}(D_{\text{task}}))]$$

1.4.9 Conclusion and Fundamental Insights

No Free Lunch Theorems

Theorem 1.4.1 (Wolpert, 1996)**.** *For any learning algorithm, its average performance over all possible target functions is equal to random guessing.*

Implication: Inductive learning requires assumptions (inductive bias) about the problem.

Key Principles of Inductive Learning

1. **Occam's Razor**: Simpler explanations are preferable (formalized via MDL)
2. **Uniform Convergence**: Empirical risk converges uniformly to true risk
3. **Stability**: Algorithm output doesn't change much with small dataset changes
4. **Compression**: Ability to compress data implies generalization

Future Directions

- **Understanding deep learning generalization**
- **Few-shot and zero-shot learning**
- **Causal inference and inductive learning**
- **Out-of-distribution generalization**
- **Federated and privacy-preserving learning**

Inductive learning represents the cornerstone of modern machine learning, providing the mathematical foundations for generalizing from finite data. While theoretical bounds often appear conservative compared to empirical success, they provide crucial guidance for model design, regularization, and understanding the fundamental limits of learning from data.

1.5 Comparative Analysis

1.5.1 Mathematical Comparison

Property	**Rote Learning**	**Inductive Learning**
Hypothesis Space	Specific examples	General functions
VC Dimension	High (memorizes any set)	Controlled by model complexity
Empirical Risk	0 (perfect on training)	May be ¿ 0
Generalization	None	Yes
Storage	O(n)	O(d) (model parameters)

Table 1.3: Comparison of learning paradigms

1.5.2 Practical Example: Learning the Sine Function

Consider learning $f(x) = \sin(x)$ from examples.

Rote learning approach:

- Memorize specific points: $\{(0, 0), (\pi/2, 1), (\pi, 0), \ldots\}$
- Cannot predict $\sin(0.5)$ if not memorized

Inductive learning approach:

- Learn parametric model: $h(x) = \sum_{k=0}^{K} a_k x^k$
- Generalize to any $x \in \mathbb{R}$
- Approximation error decreases with more training data

1.5.3 Theoretical Insights

The No Free Lunch theorem states:

Averaged over all possible data distributions, every learning algorithm has the same generalization error.

This implies:

- Inductive learning makes assumptions about the data distribution
- Rote learning makes the minimal assumption (no generalization)
- The choice depends on prior knowledge about the problem

1.6 Advanced Topics

1.6.1 Kernel Methods and Implicit Generalization

For non-linear problems, we can use kernel methods:

$$h(x) = \sum_{i=1}^{n} \alpha_i K(x, x_i)$$

where K is a kernel function measuring similarity.

1.6.2 Neural Networks as Inductive Learners

Deep learning models learn hierarchical representations:

$$h(x) = f_L(W_L f_{L-1}(W_{L-1} \cdots f_1(W_1 x + b_1) \cdots + b_{L-1}) + b_L)$$

where f_i are activation functions.

1.6.3 Bayesian Learning

Incorporating prior knowledge:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

where $P(h)$ is the prior over hypotheses.

1.7 Conclusion

1.7.1 Key Takeaways

1. **Rote learning** is suitable when:

- Exact memorization is required
- Generalization is not needed
- Dataset is small and static

2. **Inductive learning** is suitable when:
   - Generalization to unseen data is required
   - Patterns exist in the data
   - Dataset is large or noisy
3. **Hybrid approaches** (e.g., k-nearest neighbors) combine aspects of both:
   - Local memorization (rote aspect)
   - Distance-based generalization (inductive aspect)

1.7.2 Future Directions

1. **Meta-learning**: Learning to learn (inductive bias learning)
2. **Transfer learning**: Applying knowledge across domains
3. **Explainable AI**: Understanding learned representations
4. **Robust learning**: Handling distribution shifts

The choice between rote and inductive learning depends on the problem constraints, available data, and desired generalization capabilities. Modern machine learning typically employs inductive approaches with regularization to balance memorization and generalization.

Note: This document provides a comprehensive overview. For specific applications, additional considerations such as computational complexity, data quality, and domain knowledge should be taken into account.

1.8 Reinforcement Learning: Mathematical Foundations

1.8.1 Markov Decision Process (MDP) Formulation

A reinforcement learning problem is typically modeled as an MDP defined by:

$$\langle S, A, P, R, \gamma \rangle$$

where:

- S: Set of states
- A: Set of actions
- $P(s'|s,a)$: Transition probability
- $R(s,a)$: Reward function
- $\gamma \in [0,1]$: Discount factor

The goal is to find a policy $\pi : S \to A$ that maximizes the expected cumulative reward:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \middle| s_0 = s\right]$$

1.8.2 Bellman Optimality Equations

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')\right]$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

Example Problem 3: Consider a simple MDP with 2 states $S = \{s_1, s_2\}$, actions $A = \{a_1, a_2\}$, deterministic transitions, rewards $R(s_1,a_1) = 5$, $R(s_1,a_2) = 10$, $R(s_2,a_1) = -1$, $R(s_2,a_2) = 2$, and discount factor $\gamma = 0.9$. Find the optimal value function.

Solution:

Assume transitions: $s_1 \xrightarrow{a_1} s_2, \quad s_1 \xrightarrow{a_2} s_1$

$s_2 \xrightarrow{a_1} s_1, \quad s_2 \xrightarrow{a_2} s_2$

Bellman equations:

$$V^*(s_1) = \max\{5 + 0.9V^*(s_2), \quad 10 + 0.9V^*(s_1)\}$$

$$V^*(s_2) = \max\{-1 + 0.9V^*(s_1), \quad 2 + 0.9V^*(s_2)\}$$

Solving:

From $V^*(s_1) = 10 + 0.9V^*(s_1) \Rightarrow V^*(s_1) = 100$

Then $V^*(s_2) = \max\{-1 + 0.9 \times 100, \quad 2 + 0.9V^*(s_2)\}$

$= \max\{89, \quad 2 + 0.9V^*(s_2)\}$

Thus $V^*(s_2) = 89$

1.9 Types of Data in Machine Learning

1.9.1 Mathematical Characterization of Data Types

Numerical (Continuous) Data

Mathematical Representation:

$$\mathbf{x} \in \mathbb{R}^d$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is a d-dimensional real vector.

Properties:

- Forms a **metric space** $(\mathbb{R}^d, d)$ with Euclidean distance:

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

- Supports **linear operations**: addition and scalar multiplication
- Has **topological properties**: open sets, continuity, differentiability

Statistical Measures:

- **Mean**: $\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$
- **Variance**: $\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2$
- **Covariance Matrix**: $\Sigma = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$

Example Problem 1.5.1: Given two data points in $\mathbb{R}^3$: $\mathbf{a} = (1, 2, 3)$ and $\mathbf{b} = (4, 5, 6)$, calculate their Euclidean distance and cosine similarity.

Solution:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(4-1)^2 + (5-2)^2 + (6-3)^2} = \sqrt{9 + 9 + 9} = \sqrt{27} = 3\sqrt{3} \approx 5.196$$

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \frac{1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6}{\sqrt{1^2 + 2^2 + 3^2} \sqrt{4^2 + 5^2 + 6^2}} = \frac{32}{\sqrt{14}\sqrt{77}} \approx 0.974$$

Categorical (Nominal) Data

Mathematical Representation:

$$x \in C = \{c_1, c_2, \ldots, c_k\}$$

where C is a finite set of k distinct categories.

Properties:

- **Set structure**: $(C, =)$ with equality relation only
- No intrinsic ordering: $c_i \not< c_j$ unless artificially imposed
- **Cardinality**: $|C| = k$

Mathematical Operations:

- **One-hot encoding**: Map c_i to standard basis vector $\mathbf{e}_i \in \{0, 1\}^k$

$$\text{encode}(c_i) = (0, \ldots, 0, 1, 0, \ldots, 0)$$

- **Hamming distance**: For one-hot encoded vectors $\mathbf{u}, \mathbf{v}$:

$$d_H(\mathbf{u}, \mathbf{v}) = \sum_{j=1}^{k} \mathbb{I}(u_j \neq v_j)$$

where $\mathbb{I}$ is the indicator function

Statistical Measures:

- **Mode**: $\text{mode}(C) = \arg\max_{c \in C} \text{freq}(c)$
- **Probability distribution**: $p_i = P(X = c_i)$ with $\sum_{i=1}^{k} p_i = 1$

Example Problem 1.5.2: For categories $C = \{\text{Red}, \text{Green}, \text{Blue}\}$ with one-hot encoding: Red = (1,0,0), Green = (0,1,0), Blue = (0,0,1). Calculate the Hamming distance between Red and Blue.

Solution:

$$d_H(\text{Red}, \text{Blue}) = \sum_{j=1}^{3} \mathbb{I}(u_j \neq v_j) = \mathbb{I}(1 \neq 0) + \mathbb{I}(0 \neq 0) + \mathbb{I}(0 \neq 1) = 1 + 0 + 1 = 2$$

Ordinal Data

Mathematical Representation:

$$x \in \{r_1 < r_2 < \ldots < r_k\}$$

where $\{r_i\}$ forms a totally ordered set.

Properties:

- **Partial order**: $(R, \leq)$ where $r_i \leq r_j$ for $i \leq j$
- **Transitivity**: $r_i \leq r_j$ and $r_j \leq r_k$ implies $r_i \leq r_k$
- Intervals may not have consistent mathematical meaning

Mathematical Operations:

- **Rank transformation**: Map r_i to integer $i \in \{1, 2, \ldots, k\}$
- **Kendall's tau** for measuring rank correlation:

$$\tau = \frac{(\text{number of concordant pairs}) - (\text{number of discordant pairs})}{\binom{n}{2}}$$

Statistical Measures:

- **Median**: $m = \begin{cases} r_{(k+1)/2} & \text{if } k \text{ is odd} \\ \frac{1}{2}(r_{k/2} + r_{k/2+1}) & \text{if } k \text{ is even} \end{cases}$
- **Quartiles**: Q1 = 25th percentile, Q2 = median, Q3 = 75th percentile

Text Data

Mathematical Representation:

$$x \in \Sigma^* \quad \text{(set of all finite strings over alphabet } \Sigma)$$

Properties:

- **Formal language**: $L \subseteq \Sigma^*$
- **String operations**: concatenation, substring, prefix, suffix
- **Kleene star**: $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$

Vector Space Models:

- **Bag-of-words**: Represent document d as vector $\mathbf{v} \in \mathbb{N}^{|V|}$
- **TF-IDF**: For term t in document d:

$$\text{tfidf}(t, d) = \text{tf}(t, d) \times \log\left(\frac{N}{\text{df}(t)}\right)$$

where N = total documents, $\text{df}(t)$ = document frequency

Note: TF-IDF stands for Term Frequency-Inverse Document Frequency, a statistical measure used in Natural Language Processing (NLP) and information retrieval to evaluate a word's importance in a document relative to a larger collection (corpus) of documents, highlighting words frequent in one document but rare overall. It combines two parts: Term Frequency (TF) (how often a word appears in a document) and Inverse Document Frequency (IDF) (how rare the word is across all documents).

Similarity Measures:

- **Cosine similarity** for TF-IDF vectors:

$$\text{sim}(d_1, d_2) = \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\|\|\mathbf{v}_2\|}$$

- **Jaccard similarity** for sets of words:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Example Problem 1.5.3: Given two documents with TF-IDF vectors: $\mathbf{v}_1 = (0.5, 0.8, 0.3)$, $\mathbf{v}_2 = (0.2, 0.9, 0.4)$. Calculate their cosine similarity.

Solution:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = 0.5 \times 0.2 + 0.8 \times 0.9 + 0.3 \times 0.4 = 0.1 + 0.72 + 0.12 = 0.94$$

$$\|\mathbf{v}_1\| = \sqrt{0.5^2 + 0.8^2 + 0.3^2} = \sqrt{0.25 + 0.64 + 0.09} = \sqrt{0.98} \approx 0.9899$$

$$\|\mathbf{v}_2\| = \sqrt{0.2^2 + 0.9^2 + 0.4^2} = \sqrt{0.04 + 0.81 + 0.16} = \sqrt{1.01} \approx 1.0050$$

$$\text{cosine similarity} = \frac{0.94}{0.9899 \times 1.0050} \approx 0.943$$

Time Series Data

Mathematical Representation:

$$x = \{x_t\}_{t=1}^{T}, \quad x_t \in \mathbb{R}^d$$

Properties:

- **Sequence**: $x : \{1, 2, \ldots, T\} \to \mathbb{R}^d$
- **Temporal dependency**: x_t may depend on $\{x_{t-1}, x_{t-2}, \ldots\}$
- **Stationarity**: Statistical properties constant over time

Mathematical Operations:

- **Autocorrelation function**:

$$\rho(\tau) = \frac{\mathbb{E}[(x_t - \mu)(x_{t+\tau} - \mu)]}{\sigma^2}$$

- **Fourier transform**:

$$X(f) = \sum_{t=0}^{T-1} x_t e^{-i2\pi ft/T}$$

- **Lag operator**: $L(x_t) = x_{t-1}$

Time Series Models:

- **AR(p)**: Autoregressive model of order p:

$$x_t = c + \sum_{i=1}^{p} \phi_i x_{t-i} + \epsilon_t$$

- **MA(q)**: Moving average model of order q:

$$x_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

Graph Data

Mathematical Representation:

$$G = (V, E) \quad \text{where } V = \{v_1, \ldots, v_n\}, \quad E \subseteq V \times V$$

Properties:

- **Adjacency matrix**: $A \in \{0,1\}^{n \times n}$ with $A_{ij} = 1$ if $(v_i, v_j) \in E$
- **Degree matrix**: $D = \text{diag}(d_1, \ldots, d_n)$ where $d_i = \sum_j A_{ij}$
- **Laplacian matrix**: $L = D - A$

Graph Measures:

- **Degree centrality**: $C_D(v) = \frac{\deg(v)}{n-1}$
- **Betweenness centrality**:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  where σ_{st} = number of shortest paths from s to t

- **Clustering coefficient**:

$$C_i = \frac{2|\{e_{jk}\}|}{k_i(k_i - 1)} \quad \text{for } v_j, v_k \in N(i)$$

Example Problem 1.5.4: Given graph G with 4 vertices and edges: (1,2), (1,3), (2,3), (3,4). Construct the adjacency matrix and calculate vertex degrees.

Solution:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Degrees: $\deg(1) = 2$, $\deg(2) = 2$, $\deg(3) = 3$, $\deg(4) = 1$

Table 1.4: Mathematical Characterization of Data Types

Data Type	**Mathematical Space**	**Key Properties**	**Common Metrics**
Numerical	$(\mathbb{R}^d, \|\cdot\|)$	Metric space, linearity	Euclidean distance, cosine similarity
Categorical	$(C, =)$	Finite set, no ordering	Hamming distance, Jaccard index
Ordinal	$(R, \leq)$	Totally ordered set	Spearman's rho, Kendall's tau
Text	$(\Sigma^*, \cdot)$	Formal language, strings	Cosine similarity, edit distance
Time Series	$\mathbb{R}^d \times \mathbb{Z}^+$	Temporal dependency	Autocorrelation, DTW distance
Graph	(V, E)	Network structure	Shortest path, clustering coefficient

Summary Table of Mathematical Characterizations

Mathematical Implications for ML

Distance Function Selection:

$$\text{Choose } d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+ \text{ based on data properties}$$

Feature Space Embedding: For data $x \in \mathcal{X}$, find mapping $\phi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space with inner product:

$$\langle \phi(x), \phi(y) \rangle = k(x, y)$$

where k is a kernel function satisfying Mercer's condition.

Theorem (Representer Theorem): For optimization problems of the form:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2$$

the optimal solution has the form:

$$f^*(\cdot) = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot)$$

Example Problem 1.5.5: Prove that for numerical data, the Euclidean distance satisfies the triangle inequality.

Solution: For $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^d$, using Minkowski inequality:

$$d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\| = \|(\mathbf{x} - \mathbf{y}) + (\mathbf{y} - \mathbf{z})\| \leq \|\mathbf{x} - \mathbf{y}\| + \|\mathbf{y} - \mathbf{z}\| = d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$$

This mathematical characterization provides the foundation for selecting appropriate algorithms, distance metrics, and transformation techniques based on data type properties.

Table 1.5: Mathematical Representation of Data Types

Data Type	**Mathematical Representation**	**Properties**	**Example Operations**
Numerical	$x \in \mathbb{R}^d$	Metric space, linearity	Addition, multiplication
Categorical	$x \in \{c_1, c_2, \ldots, c_k\}$	Finite set, no ordering	Equality test, mode
Ordinal	$x \in \{r_1 < r_2 < \ldots < r_k\}$	Partial order	Median, percentile
Text	$x \in \Sigma^*$ (strings)	Sequence, language model	Edit distance, TF-IDF
Time Series	$x = \{x_t\}_{t=1}^T$	Temporal dependency	Autocorrelation, Fourier
Graph	$G = (V, E)$	Network structure	Shortest path, clustering

1.9.2 Data Transformation Techniques

- **Normalization**:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- **Standardization**:

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

- **One-Hot Encoding** for categorical data:

$$\text{If } x \in \{A, B, C\}, \text{ then } x_A = [1, 0, 0], x_B = [0, 1, 0], x_C = [0, 0, 1]$$

- **TF-IDF for text**:

$$\text{tfidf}(t, d) = \text{tf}(t, d) \times \log\left(\frac{N}{\text{df}(t)}\right)$$

1.10 Stages in Machine Learning Pipeline

1.10.1 Formal Mathematical Framework

The complete ML pipeline can be represented as:

$$\mathcal{M} : \mathcal{X} \to \mathcal{Y}$$

where learning involves:

$$h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h, D) + \lambda \Omega(h)$$

with:

- $\mathcal{H}$: Hypothesis space
- $\mathcal{L}$: Loss function
- Ω: Regularization term
- λ: Regularization parameter

1.10.2 Detailed Stage-by-Stage Analysis

1. **Data Acquisition**:
$$D = \{(x_i, y_i)\}_{i=1}^{n} \sim P_{data}(x, y)$$
2. **Feature Engineering**: Transform raw features $\phi : \mathcal{X} \to \mathcal{Z}$
3. **Model Selection**: Choose $h \in \mathcal{H}$ based on bias-variance tradeoff:
$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$
4. **Model Learning**: Solve optimization problem:
$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h_w(x_i)) + \lambda R(w)$$
5. **Model Evaluation**: Use metrics like:
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$
6. **Model Prediction**: For new x_{new}: $\hat{y} = h^*(x_{new})$

Example Problem 4: Given a dataset with bias-variance decomposition: $\text{Bias}^2 = 0.04$, Variance $= 0.09$, and irreducible error $= 0.02$. Calculate the expected test error and suggest whether to increase or decrease model complexity.

Solution:

$$\begin{aligned} \text{Expected Error} &= \text{Bias}^2 + \text{Variance} + \text{Irreducible Error} \\ &= 0.04 + 0.09 + 0.02 = 0.15 \end{aligned}$$

$$\text{Since Variance} > \text{Bias}^2 (0.09 > 0.04), \text{ the model is overfitting.}$$

Recommendation: Decrease model complexity to reduce variance.

1.11 Mathematical Problems and Solutions

1.11.1 Problem Set 1: Basic Calculations

1. **Problem**: Given a dataset with features normalized using z-score, with original mean $\mu = 50$ and standard deviation $\sigma = 10$, what is the normalized value for $x = 65$?

   Solution:

   $$z = \frac{x - \mu}{\sigma} = \frac{65 - 50}{10} = 1.5$$

2. **Problem**: Calculate the entropy of a binary classification problem with class distribution [0.7, 0.3].

   Solution:

   $$H = -\sum p_i \log_2 p_i = -[0.7 \log_2(0.7) + 0.3 \log_2(0.3)] \approx 0.881$$

3. **Problem**: For linear regression with MSE loss, derive the gradient with respect to weights.

   Solution:

   $$L(w) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

   $$\nabla_w L(w) = -\frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i) x_i$$

1.12 Multiple Choice Questions (MCQs)

1.12.1 Knowledge Level Questions (Remembering)

1. Who coined the term "Machine Learning"?

   (a) Alan Turing
   (b) **Arthur Samuel** (Correct)
   (c) Frank Rosenblatt
   (d) John McCarthy

2. Which of the following is NOT a type of machine learning?

   (a) Supervised Learning
   (b) Unsupervised Learning
   (c) Reinforcement Learning
   (d) **Deterministic Learning** (Correct)

1.12.2 Comprehension Level Questions (Understanding)

3. In rote learning, if a model encounters an input not in the training set, it will:

   (a) Generalize from similar examples

   (b) **Fail to produce output** (Correct)

   (c) Make a random guess

   (d) Ask for human intervention

4. The VC dimension measures:

   (a) Computation time

   (b) **Model complexity and capacity** (Correct)

   (c) Data dimensionality

   (d) Training accuracy

1.12.3 Application Level Questions (Applying)

5. Given normalized data point z = 1.5, with original $\mu = 100$, $\sigma = 20$, the original value is:

   (a) 70

   (b) 100

   (c) **130** (Correct)

   (d) 150

   Calculation: $x = z\sigma + \mu = 1.5 \times 20 + 100 = 130$

6. For a binary classifier with confusion matrix: TP=80, TN=70, FP=30, FN=20, the precision is:

   (a) **0.73** (Correct)

   (b) 0.80

   (c) 0.67

   (d) 0.75

   Calculation: Precision = TP/(TP+FP) = 80/(80+30) = 80/110 $\approx$ 0.73

1.12.4 Analysis Level Questions (Analyzing)

7. A model has high bias and low variance. This indicates:

   (a) Overfitting

   (b) **Underfitting** (Correct)

   (c) Good generalization

   (d) Perfect fit

8. In reinforcement learning, a discount factor of 0 means:

   (a) **Only immediate rewards matter** (Correct)

   (b) Future rewards are equally important

   (c) The agent is myopic

   (d) The agent considers infinite horizon

1.12.5 Synthesis Level Questions (Evaluating)

9. Given two models: Model A (Accuracy=0.85, Training time=10s), Model B (Accuracy=0.88, Training time=100s). For a real-time application, which is better?

   (a) **Model A** (Correct)

   (b) Model B

   (c) Both equally good

   (d) Cannot determine

   Reasoning: For real-time applications, training time is crucial. The 3% accuracy improvement doesn't justify 10× training time.

10. If dataset size increases from 1000 to 10000 samples, and model error decreases from 0.15 to 0.12, this demonstrates:

    (a) Overfitting

    (b) Underfitting

    (c) **The law of large numbers** (Correct)

    (d) Curse of dimensionality

1.13 Essay Questions and Solutions

1.13.1 Long Answer Questions

1. **Question**: Discuss the bias-variance tradeoff in machine learning with mathematical formulation and practical implications.

Solution: The bias-variance tradeoff is fundamental in ML. Mathematically:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x)) + \sigma^2$$

where:

- $\text{Bias}^2 = [\mathbb{E}[\hat{f}(x)] - f(x)]^2$
- $\text{Variance} = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$
- σ^2: Irreducible error

Practical Implications:

- High bias: Underfitting, model too simple
- High variance: Overfitting, model too complex
- Optimal tradeoff: Minimize total error
- Regularization helps control variance
- More data typically reduces variance

2. **Question**: Compare and contrast rote learning with inductive learning, providing mathematical formulations and real-world examples.

   Solution: **Rote Learning**:

$$\hat{y} = \begin{cases} y_i & \text{if } \exists i : x = x_i \\ \text{undefined} & \text{otherwise} \end{cases}$$

   Example: Multiplication tables, exact pattern matching.

   Inductive Learning:

$$h^* = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(y_i, h(x_i))$$

   Generalization bound (PAC learning):

$$P(\text{error}(h) \leq \epsilon) \geq 1 - \delta$$

   Comparison:

   - Rote: No generalization, exact matching required
   - Inductive: Generalizes, handles unseen data
   - Rote: Memory-intensive, fast retrieval
   - Inductive: Learning-intensive, flexible

3. **Question**: Derive the Bellman optimality equation for reinforcement learning and explain its significance.

   Solution: **Derivation**: Starting from value function definition:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right]$$

   By law of total expectation:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[r_0 + \gamma \sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| s_0 = s\right]$$

$$= \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V^{\pi}(s')]$$

   Optimal value function:

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V^*(s')]$$

   Significance:

   - Provides recursive formulation
   - Enables dynamic programming solutions
   - Foundation for Q-learning
   - Guarantees existence of optimal policy

1.14 Applications and Case Studies

1.14.1 Real-World Applications

1. **Healthcare**: Disease prediction using patient data

$$P(\text{Disease}|\text{Symptoms}) = \frac{P(\text{Symptoms}|\text{Disease})P(\text{Disease})}{P(\text{Symptoms})}$$

2. **Finance**: Credit scoring

$$\text{Credit Score} = w_1 \cdot \text{Income} + w_2 \cdot \text{Debt} + \cdots + b$$

3. **Recommendation Systems**:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

   where $\hat{r}_{ui}$ is predicted rating for user u and item i.

4. **Natural Language Processing**: Word embedding using Word2Vec:

$$\max \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log P(w_{t+j}|w_t)$$

1.14.2 Case Study: Email Spam Detection

Problem: Classify emails as spam or not spam.

Solution Approach:

1. Feature extraction: Bag of words, TF-IDF
2. Model: Naive Bayes classifier
3. Training: Maximum likelihood estimation
4. Evaluation: Precision, recall, F1-score

Mathematical Formulation:

$$P(\text{Spam}|\text{Email}) = \frac{P(\text{Email}|\text{Spam})P(\text{Spam})}{P(\text{Email})}$$

$$P(\text{Email}|\text{Spam}) = \prod_{i=1}^{n} P(\text{Word}_i|\text{Spam})$$

1.15 Summary and Key Takeaways

1.15.1 Key Mathematical Concepts

- **Learning Framework**: $h^* = \arg\min_{h \in \mathcal{H}} \mathcal{L}(h, D)$
- **Bias-Variance Tradeoff**: $\text{Error} = \text{Bias}^2 + \text{Variance} + \sigma^2$
- **Regularization**: $J(w) = L(w) + \lambda R(w)$
- **Probability Foundations**: Bayes theorem, maximum likelihood
- **Optimization**: Gradient descent, convex optimization

1.15.2 Important Formulas

Normalization: $x' = \dfrac{x - x_{min}}{x_{max} - x_{min}}$

Standardization: $z = \dfrac{x - \mu}{\sigma}$

Entropy: $H(X) = -\sum p(x) \log p(x)$

MSE: $\dfrac{1}{n} \sum (y_i - \hat{y}_i)^2$

Bellman Equation: $V(s) = \max_a [R(s, a) + \gamma \sum P(s'|s, a)V(s')]$

1.15.3 Practical Guidelines

1. Always start with simple models
2. Use cross-validation for model selection
3. Monitor bias and variance
4. Regularize to prevent overfitting
5. Feature engineering is crucial

Chapter 2

Nearest Neighbour-Based Models

2.1 Introduction to Proximity Measures

Proximity measures are fundamental tools in machine learning that quantify how similar or dissimilar two data points are. They serve as the foundation for many algorithms, particularly those based on the nearest neighbor principle. The choice of proximity measure significantly impacts algorithm performance and should align with data characteristics and problem domain.

2.1.1 Types of Proximity Measures

- **Distance Measures**: Quantify dissimilarity between objects. Smaller distances indicate greater similarity.
- **Similarity Measures**: Quantify likeness between objects. Higher values indicate greater similarity.
- **Correlation Measures**: Assess linear or monotonic relationships between variables.

2.2 Mathematical Foundations of Distance Measures

2.2.1 Formal Definition of a Metric

A distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$ is called a metric if it satisfies the following properties for all $x, y, z \in \mathcal{X}$:

1. **Non-negativity**: $d(x, y) \geq 0$
2. **Identity of Indiscernibles**: $d(x, y) = 0 \iff x = y$
3. **Symmetry**: $d(x, y) = d(y, x)$
4. **Triangle Inequality**: $d(x, z) \leq d(x, y) + d(y, z)$

These properties ensure that the distance measure behaves intuitively and supports geometric reasoning in the feature space.

2.2.2 Common Distance Measures

Minkowski Family of Distances

The Minkowski distance of order p provides a general framework:

$$d_p(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Table 2.1: Minkowski Family Distance Measures

Distance	**Formula** (p value)	**Geometric Interpretation**
Manhattan (L1)	$d_1(x, y) = \sum_{i=1}^{n} \|x_i - y_i\|$	Sum of absolute differences; grid path distance
Euclidean (L2)	$d_2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$	Straight-line distance in Euclidean space
Chebyshev (L∞)	$d_\infty(x, y) = \max_i \|x_i - y_i\|$	Maximum coordinate difference; chessboard distance
General Minkowski	$d_p(x, y) = (\sum \|x_i - y_i\|^p)^{1/p}$	Generalized distance for $p \geq 1$

Mathematical Properties of Lp Norms

For vectors $x, y \in \mathbb{R}^n$ and $1 \leq p \leq q \leq \infty$:

$$\|x\|_q \leq \|x\|_p \leq n^{\frac{1}{p} - \frac{1}{q}} \|x\|_q \quad \text{(Hölder's inequality)}$$
$$d_\infty(x, y) \leq d_2(x, y) \leq d_1(x, y) \leq n \cdot d_\infty(x, y)$$

2.2.3 Specialized Distance Measures

Mahalanobis Distance

Accounts for feature correlations and scaling:

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

where S is the covariance matrix of the data.

For a 2D point $x = [x_1, x_2]^T$ with covariance matrix $S = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$:

$$d_M(x, y) = \sqrt{\frac{(x_1 - y_1)^2}{\sigma_1^2} - \frac{2\rho(x_1 - y_1)(x_2 - y_2)}{\sigma_1\sigma_2} + \frac{(x_2 - y_2)^2}{\sigma_2^2}}$$

where ρ is the correlation coefficient.

Cosine Distance

Measures the angular difference between vectors:

$$d_{\cos}(x, y) = 1 - \frac{x \cdot y}{\|x\|\|y\|} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

2.2.4 Theoretical Analysis of Distance Measures

Isometry and Invariance Properties

Definition 2.2.1 (Isometry)**.** *A distance function d is isometry-invariant if for any isometry T (distance-preserving transformation):*

$$d(T(x), T(y)) = d(x, y)$$

- Euclidean distance is invariant under orthogonal transformations (rotations, reflections)
- Manhattan distance is invariant under axis-aligned translations and reflections
- Cosine distance is invariant under scaling: $d_{\cos}(\alpha x, \beta y) = d_{\cos}(x, y)$ for $\alpha, \beta > 0$

Curse of Dimensionality Analysis

For high-dimensional spaces ($d \to \infty$), distance measures exhibit counterintuitive behavior:

Theorem 2.2.1 (Distance Concentration)**.** *For points uniformly distributed in $[0, 1]^d$, as $d \to \infty$:*

$$\frac{\max \textit{distance} - \min \textit{distance}}{\min \textit{distance}} \to 0$$

This implies that in high dimensions, all points become approximately equidistant, making distance-based algorithms less effective.

2.3 Similarity Measures and Correlation Functions

2.3.1 Non-Metric Similarity Functions

Cosine Similarity

$$\text{cosine}(x, y) = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

Properties:

- Range: $[-1, 1]$ where 1 indicates identical orientation
- Scale-invariant: $\text{cosine}(\alpha x, \beta y) = \text{cosine}(x, y)$ for $\alpha, \beta > 0$
- Related to Euclidean distance: $\|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2\|x\|\|y\|\text{cosine}(x, y)$

Jaccard Similarity

For sets A and B:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Used for binary data or when presence/absence is more important than magnitude.

Pearson Correlation Coefficient

Measures linear correlation between variables:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

2.3.2 Proximity Between Binary Patterns

For binary vectors, we define a contingency table:

Table 2.2: Contingency Table for Binary Vectors

	Vector y		
	1	0	Total
Vector x			
1	a (both 1)	b $(x = 1, y = 0)$	$a + b$
0	c $(x = 0, y = 1)$	d (both 0)	$c + d$
Total	$a + c$	$b + d$	$n = a + b + c + d$

Binary Similarity Coefficients

Table 2.3: Common Binary Similarity Coefficients

Coefficient	**Formula**	**Emphasis**
Simple Matching	$(a + d)/n$	Both presence and absence
Jaccard	$a/(a + b + c)$	Positive matches only
Dice/Sørensen	$2a/(2a + b + c)$	Positive matches, harmonic mean
Rogers-Tanimoto	$(a + d)/(a + d + 2(b + c))$	Weighted mismatches
Hamming Distance	$b + c$	Number of differing positions

2.3.3 Mathematical Properties of Similarity Measures

Definition 2.3.1 (Positive Definite Kernel)**.** *A similarity function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel if for any $n \in \mathbb{N}$ and any $x_1, \ldots, x_n \in \mathcal{X}$, the Gram matrix K with $K_{ij} = k(x_i, x_j)$ is positive semi-definite.*

Properties:

- Cosine similarity is a positive definite kernel when restricted to non-negative vectors
- Linear kernel: $k(x, y) = x^T y$
- Gaussian RBF kernel: $k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$

2.4 Theoretical Foundations of Nearest Neighbor Methods

2.4.1 Mathematical Framework

Voronoi Decomposition

Given a set of points $\mathcal{P} = \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^d$, the Voronoi cell V_i associated with p_i is:

$$V_i = \{x \in \mathbb{R}^d : \|x - p_i\| \leq \|x - p_j\| \text{ for all } j \neq i\}$$

The collection $\{V_1, V_2, \ldots, V_n\}$ forms a Voronoi diagram of $\mathbb{R}^d$.

Decision Boundary Analysis

For two-class classification with 1-NN, the decision boundary consists of points equidistant to the nearest points of different classes:

$$\mathcal{B} = \{x \in \mathbb{R}^d : \min_{i:y_i=1} \|x - x_i\| = \min_{j:y_j=2} \|x - x_j\|\}$$

2.4.2 Convergence Properties

Universal Consistency of k-NN

Theorem 2.4.1 (Stone's Theorem, 1977)**.** *Under mild conditions on the distance metric and with $k \to \infty$, $k/n \to 0$ as $n \to \infty$, the k-NN classifier is universally consistent:*

$$\lim_{n\to\infty} \mathbb{E}[L(f_{k\text{-}NN})] = L^*,$$

where L^* is the Bayes error rate.

Rate of Convergence

For a smooth regression function f with Hölder continuity of order α:

$$\mathbb{E}[(f_{k\text{-NN}}(x) - f(x))^2] = O\left(n^{-\frac{2\alpha}{2\alpha+d}}\right)$$

where d is the dimension of the feature space.

2.5 K-Nearest Neighbor Classifier: Mathematical Analysis

2.5.1 Formal Definition

Given training data $D = \{(x_i, y_i)\}_{i=1}^n$, the k-NN classifier for a test point x is:

$$\hat{y}(x) = \arg\max_{c \in \mathcal{C}} \sum_{i \in N_k(x)} \mathbb{I}(y_i = c)$$

where $N_k(x)$ is the set of indices of the k nearest neighbors of x.

2.5.2 Algorithm with Weighted Voting

Algorithm 1 Weighted K-Nearest Neighbors Classification

Require: Training data $D = \{(x_i, y_i)\}_{i=1}^n$, test point x, k, distance function d, smoothing parameter ϵ

Ensure: Predicted class $\hat{y}$

1: Compute distances: $d_i = d(x, x_i)$ for $i = 1, \ldots, n$
2: Find indices I of k smallest distances
3: Compute weights: $w_i = \frac{1}{d_i^2 + \epsilon}$ for $i \in I$
4: **for** each class $c \in \mathcal{C}$ **do**
5: Compute score: $\text{score}(c) = \sum_{i \in I, y_i = c} w_i$
6: **end for**
7: $\hat{y} = \arg\max_{c \in \mathcal{C}} \text{score}(c)$
8: **return** $\hat{y}$

2.5.3 Bias-Variance Analysis for k-NN

Decomposition Theorem

For k-NN regression with squared error loss, the expected prediction error at point x decomposes as:

$$\mathbb{E}[(f_{k\text{-NN}}(x) - f(x))^2] = \underbrace{\left(f(x) - \frac{1}{k} \sum_{i \in N_k(x)} f(x_i) \right)^2}_{\text{Bias}^2(x)} + \underbrace{\frac{\sigma^2}{k}}_{\text{Variance}(x)} + \sigma_\epsilon^2$$

where σ^2 is the noise variance and σ_ϵ^2 is irreducible error.

Optimal Choice of k

The optimal k balances bias and variance. For fixed n, the bias decreases with k while variance increases:

- Small k: Low bias, high variance (overfitting)
- Large k: High bias, low variance (underfitting)

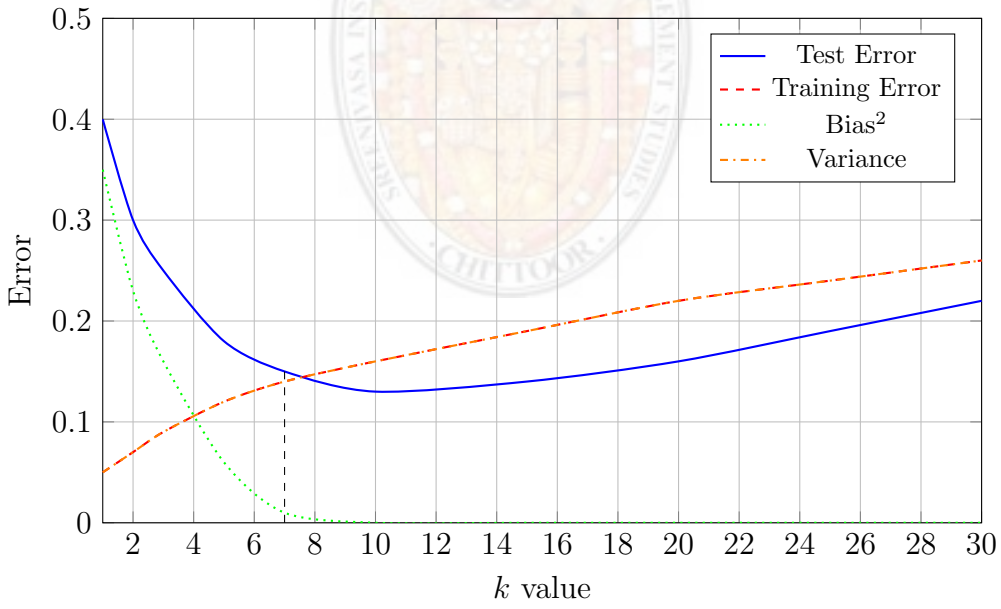


Figure 2.1: Bias-variance decomposition for different k values in k-NN

2.5.4 Computational Complexity Analysis

Naive Implementation

- **Training time**: $O(1)$ (just store the data)

- **Prediction time**: $O(nd)$ where n is dataset size, d is dimension
- **Space complexity**: $O(nd)$ (store all training points)

Optimized Data Structures

Table 2.4: Optimization Techniques for k-NN

Data Structure	Search Complexity	Best For
KD-Tree	$O(d \log n)$ average case	Low to moderate dimensions ($d < 20$)
Ball Tree	$O(d \log n)$ average case	Higher dimensions, arbitrary metrics
Locality Sensitive Hashing (LSH)	$O(1)$ approximate	Very high dimensions, approximate NN
Cover Tree	$O(\log n)$	General metric spaces

2.6 Radius-Based Nearest Neighbor Algorithm

2.6.1 Motivation and Mathematical Formulation

When data density varies significantly, fixed k may be inappropriate. The radius-based approach adapts to local density:

$$\hat{y}(x) = \arg\max_{c \in \mathcal{C}} \sum_{i: d(x, x_i) \leq r} \mathbb{I}(y_i = c)$$

where r is a fixed radius parameter.

2.6.2 Algorithm with Adaptive Radius

2.6.3 Theoretical Analysis

Probability Coverage

For a ball of radius r in $\mathbb{R}^d$, the probability that a random point falls within it is:

$$P(r) = \int_{B(x,r)} p(u)\, du$$

where $p(u)$ is the data density at point u.

Optimal Radius Selection

The optimal radius r^* minimizes the expected classification error:

$$r^* = \arg\min_r \mathbb{E}_{x \sim p(x)}[P(\hat{y}_r(x) \neq y)]$$

Algorithm 2 Adaptive Radius Nearest Neighbor Classification

Require: Training data D, test point x, initial radius r_0, distance d, minimum neighbors $m_{\min}$, maximum radius $r_{\max}$
Ensure: Predicted class $\hat{y}$
Initialize $r \leftarrow r_0$
$S \leftarrow \{x_i \in D : d(x, x_i) \leq r\}$
while $|S| < m_{\min}$ and $r \leq r_{\max}$ **do**
    $r \leftarrow r \times \alpha$ where $\alpha > 1$
    $S \leftarrow \{x_i \in D : d(x, x_i) \leq r\}$
end while
if $|S| = 0$ **then**
    return global majority class
else
    $\hat{y} \leftarrow \arg\max_{c \in \mathcal{C}} \sum_{x_i \in S} \mathbb{I}(y_i = c)$
    return $\hat{y}$
end if

2.7 K-Nearest Neighbor Regression

2.7.1 Mathematical Formulation

For regression problems, k-NN predicts the target value by averaging neighbors' values:

$$\hat{f}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i$$

2.7.2 Weighted k-NN Regression

A more sophisticated approach uses distance-weighted averaging:

$$\hat{f}(x) = \frac{\sum_{i \in N_k(x)} w_i y_i}{\sum_{i \in N_k(x)} w_i}$$

where weights can be:

- Inverse distance: $w_i = \frac{1}{d(x, x_i) + \epsilon}$
- Gaussian kernel: $w_i = \exp\left(-\frac{d(x, x_i)^2}{2h^2}\right)$
- Epanechnikov kernel: $w_i = \max(0, 1 - \frac{d(x, x_i)^2}{h^2})$

2.7.3 Convergence Analysis

Consistency Theorem

Theorem 2.7.1. *If $k \to \infty$ and $k/n \to 0$ as $n \to \infty$, and if the regression function f is continuous, then k-NN regression is consistent:*

$$\lim_{n\to\infty} \mathbb{E}[(\hat{f}_n(x) - f(x))^2] = 0$$

Rate of Convergence

For a regression function f with Lipschitz constant L:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = O\left(\left(\frac{k}{n}\right)^{2/d} + \frac{\sigma^2}{k}\right)$$

where d is dimension and σ^2 is noise variance.

2.7.4 Example: Detailed Calculation

Problem: Given training points $\{(1,3),(2,5),(3,7),(4,9)\}$, predict $f(2.5)$ using k-NN regression with $k = 2$.

Solution:

1. Compute distances:

$$\begin{aligned} d(2.5,1) &= |2.5-1| = 1.5 \\ d(2.5,2) &= |2.5-2| = 0.5 \\ d(2.5,3) &= |2.5-3| = 0.5 \\ d(2.5,4) &= |2.5-4| = 1.5 \end{aligned}$$

2. Identify 2 nearest neighbors: Points at $x = 2$ and $x = 3$ (distance 0.5 each)

3. Compute prediction: $\hat{f}(2.5) = \frac{1}{2}(5+7) = 6$

Weighted version: Using inverse distance weights with $\epsilon = 0.1$:

$$\begin{aligned} w_2 &= \frac{1}{0.5^2 + 0.1} = \frac{1}{0.25 + 0.1} = \frac{1}{0.35} \approx 2.857 \\ w_3 &= \frac{1}{0.5^2 + 0.1} = 2.857 \\ \hat{f}(2.5) &= \frac{2.857 \times 5 + 2.857 \times 7}{2.857 + 2.857} = 6 \end{aligned}$$

2.8 Advanced Nearest Neighbor Variants

2.8.1 Learning Vector Quantization (LVQ)

LVQ learns prototype vectors that represent classes. The algorithm iteratively adjusts prototypes:

Mathematical Formulation

Given prototypes m_i with labels c_i, for each training point (x, y):

- Find nearest prototype: $i^* = \arg\min_i \|x - m_i\|$
- Update rule:

$$m_{i^*} \leftarrow \begin{cases} m_{i^*} + \eta(x - m_{i^*}) & \text{if } c_{i^*} = y \\ m_{i^*} - \eta(x - m_{i^*}) & \text{if } c_{i^*} \neq y \end{cases}$$

  where η is learning rate.

2.8.2 Radial Basis Function Networks

RBF networks use distance to prototypes as features in a neural network:

$$\phi_j(x) = \exp\left(-\frac{\|x - \mu_j\|^2}{2\sigma_j^2}\right)$$

where μ_j are prototype centers and σ_j are bandwidth parameters.

The final prediction is a linear combination:

$$\hat{f}(x) = \sum_{j=1}^{m} w_j \phi_j(x) + b$$

2.9 Theoretical Limitations and Challenges

2.9.1 Curse of Dimensionality

Asymptotic Analysis

For uniformly distributed points in $[0, 1]^d$, the expected distance to the nearest neighbor is:

$$\mathbb{E}[d_{\min}] \approx \left(\frac{\Gamma(1 + d/2)}{n\pi^{d/2}}\right)^{1/d}$$

which grows rapidly with d.

Empty Space Phenomenon

In high dimensions, most of the volume of a hypercube is near the surface. For a hypercube of side length 1 in d dimensions, the fraction of volume within distance ϵ from the surface is:

$$P_{\text{surface}} = 1 - (1 - 2\epsilon)^d \approx 1 - e^{-2\epsilon d}$$

which approaches 1 as $d \to \infty$.

2.9.2 Sample Complexity Analysis

VC Dimension of k-NN

The VC dimension of 1-NN classifier with n points is:

$$d_{\text{VC}} = n$$

since it can shatter any set of n points by appropriate labeling.

For k-NN with $k > 1$, the VC dimension is more complex but generally $O(n)$.

Generalization Bounds

Using Rademacher complexity, for k-NN classifier:

$$R(\hat{f}) \leq R_n(\hat{f}) + O\left(\sqrt{\frac{n}{k}}\right)$$

where R_n is empirical risk.

2.10 Performance Evaluation of Nearest Neighbor Methods

2.10.1 Error Decomposition

For k-NN classification, the error rate decomposes as:

$$P(\text{error}) = P_{\text{Bayes}} + (1 - 2P_{\text{Bayes}}) \cdot P(\text{NN error}|\text{Bayes correct})$$

where P_{Bayes} is Bayes error rate.

2.10.2 Cross-Validation for Parameter Tuning

The optimal k can be selected via cross-validation:

$$k^* = \arg\min_k \frac{1}{m} \sum_{i=1}^{m} L(\hat{f}_k^{(-i)}(x_i), y_i)$$

where $\hat{f}_k^{(-i)}$ is the k-NN model trained without the i-th fold.

2.10.3 Confidence Intervals for k-NN

Using bootstrap, confidence intervals for k-NN predictions can be constructed:

$$\hat{f}(x) \pm t_{\alpha/2,B-1} \cdot \frac{s_B}{\sqrt{B}}$$

where s_B is standard deviation of B bootstrap predictions.

2.11 Applications and Case Studies

2.11.1 Example 1: Image Classification with k-NN

Given image features $x_i \in \mathbb{R}^{4096}$ (from CNN), k-NN can classify images:

$$\hat{y} = \arg\max_c \sum_{i \in N_k(x)} \mathbb{I}(y_i = c)$$

using Euclidean or cosine distance.

2.11.2 Example 2: Collaborative Filtering

In recommendation systems, k-NN finds similar users:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum_{v \in N_k(u)} |\text{sim}(u, v)|}$$

where $\text{sim}(u, v)$ is similarity between users u and v.

2.11.3 Example 3: Anomaly Detection

Using distance to k-th nearest neighbor as anomaly score:

$$\text{score}(x) = \max_{i \in N_k(x)} d(x, x_i)$$

Points with high scores are flagged as anomalies.

2.12 Mathematical Problems and Solutions

2.12.1 Problem Set 1: Distance Calculations

1. **Problem**: Given points $A = (1, 2, 3)$, $B = (4, 5, 6)$ in $\mathbb{R}^3$, compute:
   (a) Euclidean distance
   (b) Manhattan distance
   (c) Cosine similarity

   Solution:
   (a) $d_2(A, B) = \sqrt{(4-1)^2 + (5-2)^2 + (6-3)^2} = \sqrt{27} = 3\sqrt{3} \approx 5.196$
   (b) $d_1(A, B) = |4-1| + |5-2| + |6-3| = 9$
   (c) $\text{cosine}(A, B) = \frac{1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6}{\sqrt{1+4+9}\sqrt{16+25+36}} = \frac{32}{\sqrt{14}\sqrt{77}} \approx 0.974$

2. **Problem**: For binary vectors $x = (1, 0, 1, 0, 1)$, $y = (0, 1, 1, 0, 1)$, compute:

(a) Hamming distance

(b) Jaccard similarity

Solution:

- Contingency table: $a = 2$ (positions 3,5), $b = 1$ (position 1), $c = 1$ (position 2), $d = 1$ (position 4)
- Hamming distance: $b + c = 1 + 1 = 2$
- Jaccard similarity: $a/(a + b + c) = 2/(2 + 1 + 1) = 0.5$

2.12.2 Problem Set 2: k-NN Calculations

1. **Problem**: Given training data $\{(1, 2), (2, 4), (3, 6), (4, 8)\}$ and test point $x = 2.5$, find k-NN prediction with $k = 2$.

   Solution:

   - Distances: $d(2.5, 1) = 1.5$, $d(2.5, 2) = 0.5$, $d(2.5, 3) = 0.5$, $d(2.5, 4) = 1.5$
   - Nearest neighbors: $x = 2$ (y=4) and $x = 3$ (y=6)
   - Prediction: $\hat{y} = (4 + 6)/2 = 5$

2. **Problem**: For the same data, compute weighted k-NN prediction with inverse distance weights.

   Solution:

   - Weights: $w_2 = 1/0.5^2 = 4$, $w_3 = 1/0.5^2 = 4$
   - Prediction: $\hat{y} = (4 \times 4 + 4 \times 6)/(4 + 4) = 5$

2.13 Multiple Choice Questions

2.13.1 Knowledge Level Questions

1. Which property is NOT required for a distance metric?

   (a) Non-negativity

   (b) Symmetry

   (c) **Linearity** (Correct)

   (d) Triangle inequality

2. The curse of dimensionality primarily affects k-NN by:

   (a) Increasing computation time

   (b) **Making distances less meaningful** (Correct)

   (c) Reducing model flexibility

   (d) Increasing bias

2.13.2 Comprehension Level Questions

3. For small k in k-NN, the model typically has:

   (a) High bias, low variance
   (b) **Low bias, high variance** (Correct)
   (c) Both high bias and variance
   (d) Both low bias and variance

4. The VC dimension of 1-NN with n training points is:

   (a) $O(1)$
   (b) $O(\log n)$
   (c) $O(n)$ (Correct)
   (d) $O(2^n)$

2.13.3 Application Level Questions

5. Given points $A = (0, 0)$, $B = (3, 4)$, the Euclidean distance is:

   (a) 3
   (b) 4
   (c) 5
   (d) **5** (Correct)

   Calculation: $d = \sqrt{(3-0)^2 + (4-0)^2} = \sqrt{9+16} = \sqrt{25} = 5$

6. For k-NN classification with $k = 3$ and neighbors' classes = A, A, B, the predicted class is:

   (a) A (Correct)
   (b) B
   (c) Cannot determine
   (d) Requires distance weights

2.14 Essay Questions and Solutions

1. **Question**: Discuss the bias-variance tradeoff in k-NN regression with mathematical formulation and practical implications.

   Solution: The bias-variance decomposition for k-NN regression at point x is:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = \underbrace{(f(x) - \mathbb{E}[\hat{f}(x)])^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]}_{\text{Variance}} + \sigma^2$$

   Practical Implications:

- Small k: Low bias (fits local variations) but high variance (sensitive to noise)
- Large k: High bias (oversmooths) but low variance (stable predictions)
- Optimal k: Balances bias and variance, typically found via cross-validation
- In high dimensions: Larger k needed due to curse of dimensionality

2. **Question**: Compare and contrast Euclidean, Manhattan, and cosine distances, providing mathematical formulations and suitable applications for each.

   Solution:

   Euclidean (L2):

$$d_2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

   - Properties: Rotation invariant, isotropic
   - Applications: Physical distances, Gaussian-distributed features

   Manhattan (L1):

$$d_1(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

   - Properties: Robust to outliers, grid-like paths
   - Applications: Urban navigation, sparse data

   Cosine:

$$d_{\cos}(x, y) = 1 - \frac{x \cdot y}{\|x\|\|y\|}$$

   - Properties: Scale invariant, measures angular similarity
   - Applications: Text analysis, high-dimensional sparse data

3. **Question**: Derive the rate of convergence for k-NN regression and explain the impact of dimensionality.

   Solution: For a regression function f with Lipschitz constant L:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] \leq L^2 \mathbb{E}[\|x - x_{(k)}\|^2] + \frac{\sigma^2}{k}$$

   where $x_{(k)}$ is the k-th nearest neighbor.

   For points uniformly distributed in $[0, 1]^d$:

$$\mathbb{E}[\|x - x_{(k)}\|^2] \approx \left(\frac{k}{n}\right)^{2/d}$$

   Thus:

$$\mathbb{E}[(\hat{f}(x) - f(x))^2] = O\left(\left(\frac{k}{n}\right)^{2/d} + \frac{\sigma^2}{k}\right)$$

   Dimensionality Impact:

- As d increases, $\left(\frac{k}{n}\right)^{2/d} \to 1$ for fixed k, n
- Requires exponentially more data to maintain same accuracy
- Explains curse of dimensionality in k-NN

2.15 Summary and Key Takeaways

2.15.1 Mathematical Foundations

- **Distance metrics** must satisfy four properties: non-negativity, identity, symmetry, triangle inequality
- **Minkowski family** provides general framework: $d_p(x, y) = (\sum |x_i - y_i|^p)^{1/p}$
- **Mahalanobis distance** accounts for feature correlations: $d_M = \sqrt{(x-y)^T S^{-1}(x-y)}$
- **Cosine distance** measures angular difference: $d_{\cos} = 1 - \frac{x \cdot y}{\|x\|\|y\|}$

2.15.2 Algorithmic Properties

- **k-NN classification**: $\hat{y} = \arg\max_c \sum_{i \in N_k(x)} \mathbb{I}(y_i = c)$
- **k-NN regression**: $\hat{f}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i$
- **Bias-variance tradeoff**: Small k = low bias, high variance; Large k = high bias, low variance
- **Optimal k selection**: Typically via cross-validation

2.15.3 Theoretical Insights

- **Curse of dimensionality**: Distances become less meaningful in high dimensions
- **Universal consistency**: k-NN is consistent if $k \to \infty$, $k/n \to 0$
- **Rate of convergence**: $O(n^{-2/(2+d)})$ for smooth functions
- **VC dimension**: 1-NN has VC dimension = n

2.15.4 Practical Guidelines

1. **Feature scaling**: Normalize features before applying distance measures
2. **Distance selection**: Choose metric based on data characteristics
3. **Dimensionality reduction**: Apply PCA or other methods for high-dimensional data
4. **Parameter tuning**: Use cross-validation to select optimal k
5. **Computational optimization**: Use KD-trees, ball trees, or LSH for large datasets

2.15.5 Important Formulas

$$\text{Euclidean: } d_2(x, y) = \sqrt{\sum (x_i - y_i)^2}$$

$$\text{Manhattan: } d_1(x, y) = \sum |x_i - y_i|$$

$$\text{Cosine: } \text{sim}(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

$$\text{Mahalanobis: } d_M(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

$$\text{k-NN prediction: } \hat{y} = \arg\max_c \sum_{i \in N_k(x)} w_i \mathbb{I}(y_i = c)$$

$$\text{Weighted k-NN: } w_i = \frac{1}{d(x, x_i)^2 + \epsilon}$$

2.15.6 Future Directions

1. **Adaptive metrics**: Learning distance metrics from data
2. **Deep metric learning**: Using neural networks to learn embeddings
3. **Approximate nearest neighbors**: Scalable algorithms for massive datasets
4. **Theoretical foundations**: Better understanding of k-NN in high dimensions
5. **Hybrid approaches**: Combining k-NN with other methods for improved performance

---

Note: Nearest neighbor methods provide simple yet powerful non-parametric approaches to both classification and regression. Their strength lies in their flexibility and asymptotic optimality, though they face challenges in high dimensions and with large datasets. Understanding the mathematical foundations is crucial for proper application and extension of these methods.

Chapter 3

UNIT 3: MODELS BASED ON DECISION TREES

3.1 INTRODUCTION TO DECISION TREES

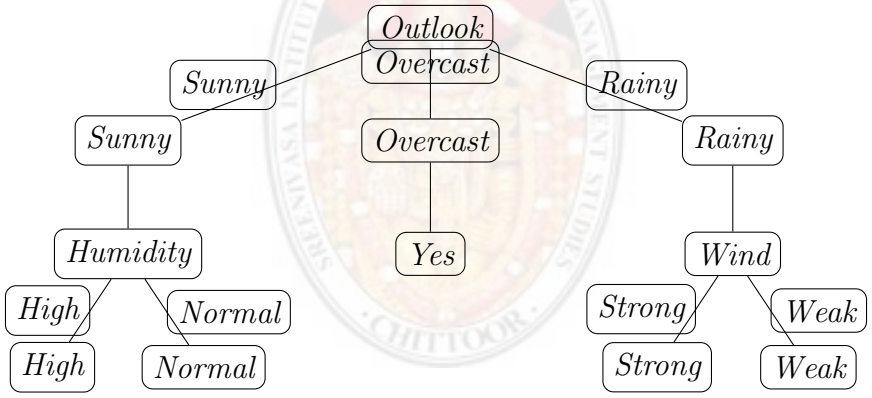
Decision trees are one of the most intuitive and interpretable machine learning algorithms. They mimic human decision-making processes by breaking down complex decisions into a series of simpler choices.

3.1.1 Basic Structure

A decision tree consists of the following components:

- **Root Node**: The topmost node representing the entire dataset
- **Internal Nodes**: Represent tests on attributes/features
- **Branches**: Represent outcomes of the tests
- **Leaf Nodes**: Represent class labels or final decisions

Example 3.1.1 (Weather Prediction Decision Tree)**.** *Consider a decision tree for deciding whether to play tennis based on weather conditions:*



At each internal node, a test is performed on an attribute, and based on the outcome, we traverse down the corresponding branch until we reach a leaf node that provides the classification.

3.1.2 Decision Tree Learning Algorithm

The basic algorithm for building decision trees (known as ID3, C4.5, or CART) follows a recursive partitioning approach:

Definition 3.1.1 (Decision Tree Building Algorithm)**.** ***Input:*** *Training dataset D with features A and class labels*
Output: *A decision tree T*

1. *Start with the entire dataset at the root node*

2. *If all instances in the current node belong to the same class, create a leaf node with that class*
3. *If no features remain to split on, create a leaf node with the majority class*
4. *Otherwise:*
   (a) *Select the "best" attribute A to split the data using an impurity measure*
   (b) *Create a branch for each possible value of A*
   (c) *Partition the data into subsets D_v where attribute A has value v*
   (d) *Recursively repeat steps 2-5 for each branch*

3.2 IMPURITY MEASURES

Impurity measures quantify how "mixed" the classes are at a node. The goal is to select splits that minimize impurity.

3.2.1 Entropy

Entropy, borrowed from information theory, measures the disorder or uncertainty in a dataset.

Definition 3.2.1 (Entropy)**.** *For a dataset D with k classes, entropy is defined as:*

$$H(D) = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

where p_i is the proportion of instances belonging to class i.

Properties of Entropy:

- Range: $0 \leq H(D) \leq \log_2(k)$
- $H(D) = 0$ when all instances belong to the same class (pure node)
- $H(D)$ is maximum when classes are equally distributed
- For binary classification with probabilities p and $(1-p)$:

$$H(D) = -p \log_2 p - (1-p) \log_2(1-p)$$

Example 3.2.1 (Computing Entropy)**.** *Consider a dataset with 10 instances: 6 positive and 4 negative.*

$$p_+ = \frac{6}{10} = 0.6, \quad p_- = \frac{4}{10} = 0.4$$

$$\begin{aligned} H(D) &= -0.6\log_2(0.6) - 0.4\log_2(0.4) \\ &= -0.6(-0.737) - 0.4(-1.322) \\ &= 0.4422 + 0.5288 = 0.971 \text{ } \textit{bits} \end{aligned}$$

This indicates moderate impurity—the dataset is not completely pure.

3.2.2 Gini Index

The Gini index, used in the CART algorithm, measures the probability of misclassifying a randomly chosen instance if it were randomly labeled according to the class distribution.

Definition 3.2.2 (Gini Index)**.** *For a dataset D with k classes, the Gini index is:*

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

where p_i is the proportion of instances belonging to class i.

Properties of Gini Index:

- Range: $0 \leq Gini(D) \leq 1 - \frac{1}{k}$
- Minimum (0) occurs when all instances belong to one class
- Maximum occurs when classes are equally distributed
- For binary classification with probabilities p and $(1-p)$:

$$Gini(D) = 2p(1-p)$$

Example 3.2.2 (Computing Gini Index)**.** *Using the same dataset (6 positive, 4 negative):*

$$Gini(D) = 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = 0.48$$

This value indicates that if we randomly assign labels according to the class distribution, we would misclassify about 48% of instances.

3.2.3 Misclassification Error

The misclassification error is the simplest impurity measure.

Definition 3.2.3 (Misclassification Error)**.** *For a dataset D with k classes:*

$$Error(D) = 1 - \max_i(p_i)$$

where p_i is the proportion of instances belonging to class i.

Example 3.2.3 (Computing Misclassification Error)**.** *For the dataset with 6 positive, 4 negative:*

$$\max(p_+, p_-) = \max(0.6, 0.4) = 0.6$$

$$Error(D) = 1 - 0.6 = 0.4$$

This means that if we always predicted the majority class, we would be wrong 40% of the time.

3.2.4 Comparison of Impurity Measures

For binary classification, we can compare all three measures:

Table 3.1: Impurity Measures for Binary Classification

Measure	**Formula**	**Range**
Entropy	$-p \log_2 p - (1-p) \log_2(1-p)$	[0, 1]
Gini	$2p(1-p)$	[0, 0.5]
Error	$1 - \max(p, 1-p)$	[0, 0.5]

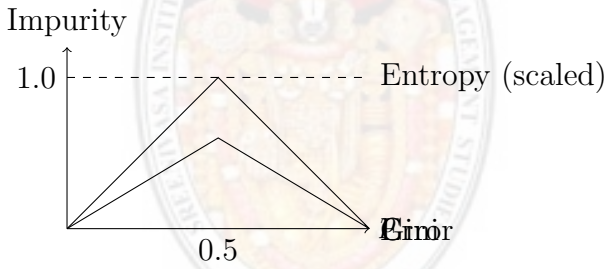


Figure 3.1: Impurity measures as functions of p (for binary classification)

Key Observations:

- Entropy is more sensitive to changes in class distribution
- Gini and Error are similar but Gini is differentiable (useful for optimization)
- Error is piecewise linear but has a flat region where it doesn't change

3.3 INFORMATION GAIN AND SPLITTING CRITERIA

3.3.1 Information Gain

Information gain measures the reduction in impurity achieved by splitting on an attribute.

Definition 3.3.1 (Information Gain)**.** *For a dataset D and an attribute A, information gain is:*

$$IG(D, A) = I(D) - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} I(D_v)$$

where I is an impurity measure (entropy, Gini, or error), and D_v is the subset of D where attribute A has value v.

The attribute with the highest information gain is selected for splitting.

3.3.2 Gain Ratio

Information gain tends to favor attributes with many values. Gain ratio addresses this bias.

Definition 3.3.2 (Gain Ratio)**.**

$$GainRatio(D, A) = \frac{IG(D, A)}{SplitInfo(D, A)}$$

where

$$SplitInfo(D, A) = -\sum_{v \in Values(A)} \frac{|D_v|}{|D|} \log_2 \left(\frac{|D_v|}{|D|} \right)$$

3.3.3 Worked Example: Building a Decision Tree

Let's build a decision tree using the "Play Tennis" dataset with entropy as the impurity measure.

Table 3.2: Play Tennis Dataset

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Step 1: Calculate Entropy of Root Node

Total instances: 14 (9 Yes, 5 No)

$$\begin{aligned} H(D) &= -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} \\ &= -0.643 \times (-0.637) - 0.357 \times (-1.485) \\ &= 0.410 + 0.530 = 0.940 \text{ bits} \end{aligned}$$

Step 2: Calculate Information Gain for Each Attribute

Attribute: Outlook

Outlook has three values: Sunny, Overcast, Rainy

$$\begin{aligned} D_{\text{Sunny}} &: 5 \text{ instances (2 Yes, 3 No)} \\ H(D_{\text{Sunny}}) &= -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} \\ &= -0.4 \times (-1.322) - 0.6 \times (-0.737) = 0.529 + 0.442 = 0.971 \end{aligned}$$

$$\begin{aligned} D_{\text{Overcast}} &: 4 \text{ instances (4 Yes, 0 No)} \\ H(D_{\text{Overcast}}) &= -\frac{4}{4}\log_2 1 - 0 = 0 \end{aligned}$$

$$\begin{aligned} D_{\text{Rainy}} &: 5 \text{ instances (3 Yes, 2 No)} \\ H(D_{\text{Rainy}}) &= -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} \\ &= -0.6 \times (-0.737) - 0.4 \times (-1.322) = 0.442 + 0.529 = 0.971 \end{aligned}$$

Weighted average entropy after splitting on Outlook:

$$\begin{aligned} H(D|\text{Outlook}) &= \frac{5}{14} \times 0.971 + \frac{4}{14} \times 0 + \frac{5}{14} \times 0.971 \\ &= 0.357 \times 0.971 + 0.286 \times 0 + 0.357 \times 0.971 \\ &= 0.347 + 0 + 0.347 = 0.694 \end{aligned}$$

Information Gain for Outlook:

$$IG(D, \text{Outlook}) = 0.940 - 0.694 = 0.246$$

Attribute: Temperature

Temperature has three values: Hot, Mild, Cool

$$D_{\text{Hot}} : 4 \text{ instances (2 Yes, 2 No)}$$
$$H(D_{\text{Hot}}) = -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = -0.5 \times (-1) - 0.5 \times (-1) = 0.5 + 0.5 = 1$$

$$D_{\text{Mild}} : 6 \text{ instances (4 Yes, 2 No)}$$
$$\begin{aligned} H(D_{\text{Mild}}) &= -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6} \\ &= -0.667 \times (-0.585) - 0.333 \times (-1.585) = 0.390 + 0.528 = 0.918 \end{aligned}$$

$$D_{\text{Cool}} : 4 \text{ instances (3 Yes, 1 No)}$$
$$\begin{aligned} H(D_{\text{Cool}}) &= -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} \\ &= -0.75 \times (-0.415) - 0.25 \times (-2) = 0.311 + 0.5 = 0.811 \end{aligned}$$

Weighted average entropy after splitting on Temperature:

$$\begin{aligned} H(D|\text{Temperature}) &= \frac{4}{14} \times 1 + \frac{6}{14} \times 0.918 + \frac{4}{14} \times 0.811 \\ &= 0.286 \times 1 + 0.429 \times 0.918 + 0.286 \times 0.811 \\ &= 0.286 + 0.394 + 0.232 = 0.912 \end{aligned}$$

Information Gain for Temperature:

$$IG(D, \text{Temperature}) = 0.940 - 0.912 = 0.028$$

Attribute: Humidity
Humidity has two values: High, Normal

$$D_{\text{High}} : 7 \text{ instances (3 Yes, 4 No)}$$
$$\begin{aligned} H(D_{\text{High}}) &= -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} \\ &= -0.429 \times (-1.222) - 0.571 \times (-0.807) = 0.524 + 0.461 = 0.985 \end{aligned}$$

$$D_{\text{Normal}} : 7 \text{ instances (6 Yes, 1 No)}$$
$$\begin{aligned} H(D_{\text{Normal}}) &= -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} \\ &= -0.857 \times (-0.222) - 0.143 \times (-2.807) = 0.190 + 0.401 = 0.591 \end{aligned}$$

Weighted average entropy after splitting on Humidity:

$$H(D|\text{Humidity}) = \frac{7}{14} \times 0.985 + \frac{7}{14} \times 0.591$$
$$= 0.5 \times 0.985 + 0.5 \times 0.591 = 0.493 + 0.296 = 0.789$$

Information Gain for Humidity:

$$IG(D, \text{Humidity}) = 0.940 - 0.789 = 0.151$$

Attribute: Wind
Wind has two values: Weak, Strong

$$D_{\text{Weak}} : 8 \text{ instances (6 Yes, 2 No)}$$
$$H(D_{\text{Weak}}) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8}$$
$$= -0.75 \times (-0.415) - 0.25 \times (-2) = 0.311 + 0.5 = 0.811$$

$$D_{\text{Strong}} : 6 \text{ instances (3 Yes, 3 No)}$$
$$H(D_{\text{Strong}}) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = -0.5 \times (-1) - 0.5 \times (-1) = 1$$

Weighted average entropy after splitting on Wind:

$$H(D|\text{Wind}) = \frac{8}{14} \times 0.811 + \frac{6}{14} \times 1$$
$$= 0.571 \times 0.811 + 0.429 \times 1 = 0.463 + 0.429 = 0.892$$

Information Gain for Wind:

$$IG(D, \text{Wind}) = 0.940 - 0.892 = 0.048$$

Step 3: Select Best Attribute for Root

Attribute	**Information Gain**
Outlook	0.246
Temperature	0.028
Humidity	0.151
Wind	0.048

Outlook has the highest information gain, so we select it as the root node.

Step 4: Build Subtrees

For Outlook = Overcast: All 4 instances are "Yes", so this branch becomes a leaf node with class "Yes".

For Outlook = Sunny: We have 5 instances: D1, D2, D8, D9, D11 (2 Yes, 3 No)
We need to find the best attribute among Temperature, Humidity, Wind for this subset.
Calculate entropy for this subset:

$$H(D_{\text{Sunny}}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.971$$

For Humidity in Sunny subset:

$$\begin{aligned} D_{\text{High}} &: 3 \text{ instances (0 Yes, 3 No)} \Rightarrow H = 0 \\ D_{\text{Normal}} &: 2 \text{ instances (2 Yes, 0 No)} \Rightarrow H = 0 \\ H(D_{\text{Sunny}}|\text{Humidity}) &= \frac{3}{5}\times 0 + \frac{2}{5}\times 0 = 0 \end{aligned}$$

$$IG(D_{\text{Sunny}}, \text{Humidity}) = 0.971 - 0 = 0.971$$

For Temperature in Sunny subset:

$$\begin{aligned} D_{\text{Hot}} &: 2 \text{ instances (0 Yes, 2 No)} \Rightarrow H = 0 \\ D_{\text{Mild}} &: 2 \text{ instances (1 Yes, 1 No)} \Rightarrow H = 1 \\ D_{\text{Cool}} &: 1 \text{ instance (1 Yes, 0 No)} \Rightarrow H = 0 \\ H(D_{\text{Sunny}}|\text{Temperature}) &= \frac{2}{5}\times 0 + \frac{2}{5}\times 1 + \frac{1}{5}\times 0 = 0.4 \end{aligned}$$

$$IG(D_{\text{Sunny}}, \text{Temperature}) = 0.971 - 0.4 = 0.571$$

For Wind in Sunny subset:

$$\begin{aligned} D_{\text{Weak}} &: 3 \text{ instances (1 Yes, 2 No)} \\ H(D_{\text{Weak}}) &= -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \\ &= -0.333\times(-1.585) - 0.667\times(-0.585) = 0.528 + 0.390 = 0.918 \\ D_{\text{Strong}} &: 2 \text{ instances (1 Yes, 1 No)} \Rightarrow H = 1 \\ H(D_{\text{Sunny}}|\text{Wind}) &= \frac{3}{5}\times 0.918 + \frac{2}{5}\times 1 = 0.551 + 0.4 = 0.951 \end{aligned}$$

$$IG(D_{\text{Sunny}}, \text{Wind}) = 0.971 - 0.951 = 0.020$$

Humidity gives the highest information gain, so we split on Humidity at this node.
For Outlook = Rainy: We have 5 instances: D4, D5, D6, D10, D14 (3 Yes, 2 No)
Entropy: $H(D_{\text{Rainy}}) = 0.971$ (same as Sunny)

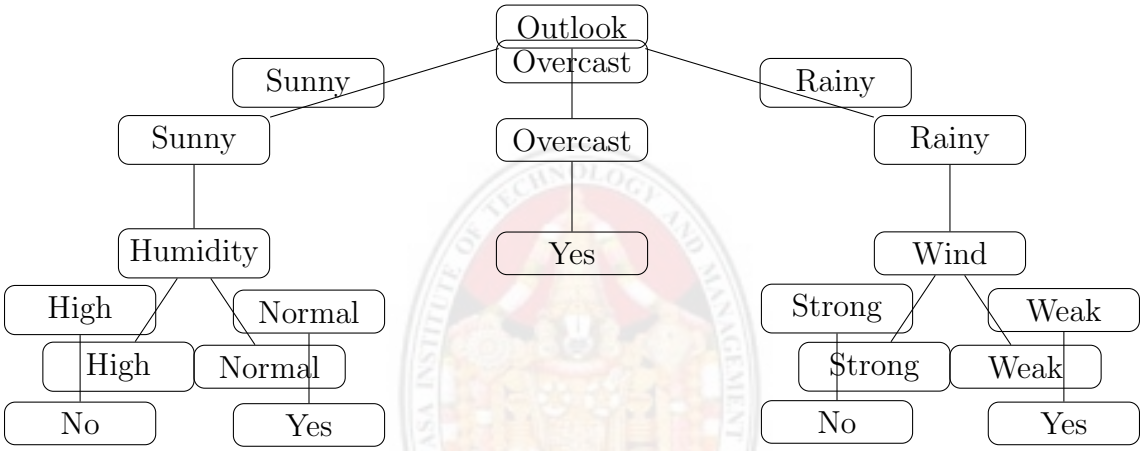
For Wind in Rainy subset:

$$D_{\text{Weak}} : 3 \text{ instances (3 Yes, 0 No)} \Rightarrow H = 0$$
$$D_{\text{Strong}} : 2 \text{ instances (0 Yes, 2 No)} \Rightarrow H = 0$$
$$H(D_{\text{Rainy}}|\text{Wind}) = 0$$

$$IG(D_{\text{Rainy}}, \text{Wind}) = 0.971 - 0 = 0.971$$

Wind perfectly separates the classes, so we split on Wind at this node.

Step 5: Final Decision Tree



3.4 PROPERTIES OF DECISION TREES

3.4.1 Advantages

1. **Interpretability**: Decision trees are easy to understand and explain
2. **No need for data preprocessing**: They handle both numerical and categorical data without scaling
3. **Non-parametric**: No assumptions about data distribution
4. **Feature selection**: Automatically selects important features
5. **Handles non-linear relationships**: Can capture complex patterns

3.4.2 Disadvantages

1. **Overfitting**: Trees can grow too deep and memorize noise
2. **Instability**: Small changes in data can lead to completely different trees

3. **Bias toward features with many values**: Information gain favors features with many distinct values
4. **Optimal split problem**: Finding the globally optimal tree is NP-hard

3.5 REGRESSION BASED ON DECISION TREES

Decision trees can also be used for regression tasks where the target variable is continuous.

3.5.1 Regression Tree Structure

In regression trees:

- Leaf nodes contain a constant value (usually the mean of target values in that node)
- Splits are chosen to minimize the variance or sum of squared errors

Definition 3.5.1 (Regression Tree Splitting Criterion)**.** *For a node with dataset D, we choose the split that minimizes:*

$$\sum_{v \in Values(A)} \sum_{x \in D_v} (y - \bar{y}_v)^2$$

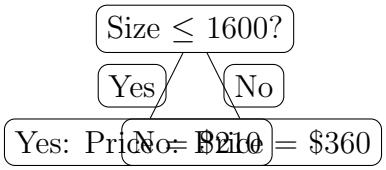
where $\bar{y}_v$ is the mean of target values in subset D_v.

3.5.2 Example of Regression Tree

Consider predicting house prices based on size:

House	Size (sq ft)	Price ($1000)
1	800	150
2	1000	180
3	1200	210
4	1400	240
5	1600	270
6	1800	300
7	2000	330
8	2200	360
9	2400	390
10	2600	420

A regression tree might split at size = 1600:



The predicted values are the averages:

$$\text{Left node average} = \frac{150 + 180 + 210 + 240 + 270}{5} = 210$$
$$\text{Right node average} = \frac{300 + 330 + 360 + 390 + 420}{5} = 360$$

3.6 BIAS-VARIANCE TRADEOFF

3.6.1 Definitions

Definition 3.6.1 (Bias)**.** *Bias is the error due to incorrect assumptions in the learning algorithm. High bias can cause underfitting.*

$$Bias[\hat{f}(x)] = \mathbb{E}[\hat{f}(x)] - f(x)$$

Definition 3.6.2 (Variance)**.** *Variance is the error due to sensitivity to fluctuations in the training set. High variance can cause overfitting.*

$$Var[\hat{f}(x)] = \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]$$

3.6.2 The Tradeoff

The expected prediction error can be decomposed as:

Theorem 3.6.1 (Bias-Variance Decomposition)**.** *For a target variable $y = f(x) + \epsilon$ with $\mathbb{E}[\epsilon] = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$:*

$$\mathbb{E}[(y - \hat{f}(x))^2] = Bias[\hat{f}(x)]^2 + Var[\hat{f}(x)] + \sigma_\epsilon^2$$

3.6.3 Decision Tree Complexity

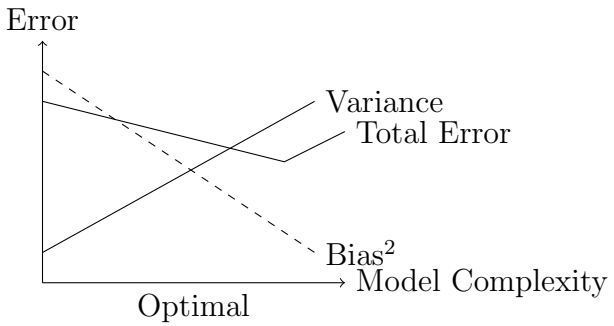


Figure 3.2: Bias-variance tradeoff in decision trees

Implications for Decision Trees:

- **Shallow trees (high bias)**: May underfit, missing important patterns

- **Deep trees (high variance)**: May overfit, capturing noise in the training data
- **Pruning**: Helps find the optimal complexity by reducing variance without increasing bias too much

3.7 RANDOM FORESTS FOR CLASSIFICATION AND REGRESSION

Random forests are an ensemble method that combines multiple decision trees to improve performance.

3.7.1 Bootstrap Aggregation (Bagging)

In Machine Learning, Bagging (short for Bootstrap Aggregating) is an ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms. It is used to reduce variance and help avoid overfitting.

Definition 3.7.1 (Bagging)**.** *Bagging (Bootstrap Aggregating) involves:*

1. *Creating B bootstrap samples from the training data (sampling with replacement)*
2. *Training a decision tree on each bootstrap sample*
3. *For classification: majority voting among trees*
4. *For regression: averaging predictions from all trees*

High-variance models (like deep decision trees) are very sensitive to the specific data they are trained on. If you change the training data slightly, the model's predictions can change dramatically. This leads to overfitting—the model learns the noise in the training data rather than the actual signal.

3.7.2 Bagging (Bootstrap Aggregating)

In Machine Learning, **Bagging** (short for **Bootstrap Aggregating**) is an ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms. It is used to reduce **variance** and help **avoid overfitting**.

The Core Problem It Solves

High-variance models (like deep decision trees) are very sensitive to the specific data they are trained on. If you change the training data slightly, the model's predictions can change dramatically. This leads to **overfitting**—the model learns the noise in the training data rather than the actual signal.

How Bagging Works

Bagging combines the concept of "Bootstrap" with "Aggregating."

The process involves three main steps:

Step 1: Bootstrapping (Creating Multiple Datasets)

- You start with one original training dataset containing, say, N samples.
- You create M new datasets (where M is the number of models you want to train).
- Each new dataset is created by **randomly sampling** N instances **with replacement** from the original dataset.
- *Note:* Because you are sampling with replacement, each new dataset will have some duplicate samples and will be missing about 37% of the original data points (these missing ones are called Out-Of-Bag samples).

Step 2: Parallel Training

- You train a separate base model (often called a "weak learner") on each of these M bootstrapped datasets.
- Crucially, these models are trained **independently and in parallel** (unlike Boosting, where models are trained sequentially).

Step 3: Aggregating

- You combine the predictions from all the trained models to make a final prediction.
- **For Regression:** Take the **average** of all the predictions.
- **For Classification:** Take a **majority vote** (the class predicted most frequently) among all the models.

Why Does This Work? (The Math)

Imagine you have M independent models, each with a variance of σ^2.

If you average these M models together, the variance of the average is:

$$\sigma^2_{\text{ensemble}} = \frac{1}{M}\sigma^2$$

However, in practice, the models are not perfectly independent because they are trained on similar (bootstrapped) data. But by introducing randomness through bootstrapping, the errors made by individual models become less correlated. When you average their predictions, the uncorrelated errors cancel each other out, resulting in a smoother, more accurate prediction.

3.7.3 Random Forest Algorithm

Random forests extend bagging by adding random feature selection at each split.

Definition 3.7.2 (Random Forest)**.** *For each tree in the forest:*

1. *Draw a bootstrap sample of size N from the training data*
2. *At each node, randomly select m features from the total p features*
3. *Choose the best split among these m features*
4. *Grow the tree to full depth (no pruning)*

The recommended value for m is $\sqrt{p}$ for classification and $p/3$ for regression.

A Classic Example: Random Forest

The most famous and widely used application of Bagging is the **Random Forest** algorithm.

- **Base Model:** Decision Trees.
- **Bagging Applied:** Each tree is trained on a different bootstrap sample of the data.
- **Extra Trick:** Random Forest adds an extra layer of randomness. Instead of looking for the best feature to split on among all features, it only looks at a random subset of features. This further decorrelates the trees.

Key Benefits

1. **Reduces Overfitting:** It smooths out the predictions, making the model generalize better to unseen data.
2. **Increases Stability:** Small changes in the input data have less impact on the final output.
3. **Out-of-Bag (OOB) Evaluation:** Because each model is trained on only about 63% of the data, the remaining 37% (OOB samples) can be used as a built-in validation set. You can evaluate the model's performance without needing a separate test set.

Summary

Bagging is a technique where you take a high-variance model, train multiple copies of it on random subsets of the data (created via bootstrapping), and then average their outputs to create a single, more robust, and accurate model.

3.7.4 Out-of-Bag Error

Out-of-bag (OOB) error provides an unbiased estimate of the generalization error without needing a separate validation set.

Definition 3.7.3 (OOB Error)**.** *For each training instance, predict its class using only trees that were trained on bootstrap samples not containing that instance. The OOB error is the misclassification rate across all instances.*

Why OOB works:

- Each bootstrap sample contains about 63.2% of the original data
- About 36.8% of instances are left out (out-of-bag)
- These OOB instances serve as a natural validation set

3.7.5 Feature Importance

Random forests provide measures of feature importance:

1. **Mean Decrease in Impurity**: For each feature, sum the decreases in impurity (e.g., Gini) over all splits where that feature was used, weighted by the number of instances at each node
2. **Mean Decrease in Accuracy**: Permute the values of a feature in OOB instances and measure the drop in prediction accuracy

3.7.6 Convergence Property

Theorem 3.7.1 (Random Forest Convergence)**.** *As the number of trees increases, the generalization error of random forests converges to a limit, and random forests do not overfit as more trees are added.*

3.8 THE BAYES CLASSIFIER

3.8.1 Introduction to Bayes Classifier

The Bayes classifier is a probabilistic approach to classification that uses Bayes' theorem to compute the probability of each class given the observed features.

Definition 3.8.1 (Bayes Classifier)**.** *Given a feature vector x and classes $C_1, C_2, \ldots, C_k$, the Bayes classifier assigns x to class C_i that maximizes the posterior probability:*

$$P(C_i|x) = \frac{P(x|C_i)P(C_i)}{P(x)}$$

3.8.2 Bayes' Rule and Inference

Bayes' theorem provides a way to update probabilities based on evidence:

Theorem 3.8.1 (Bayes' Theorem)**.**

$$P(C_i|x) = \frac{P(x|C_i)P(C_i)}{\sum_{j=1}^{k} P(x|C_j)P(C_j)}$$

where:

- $P(C_i|x)$ *is the* ***posterior probability***
- $P(x|C_i)$ *is the* ***likelihood***
- $P(C_i)$ *is the* ***prior probability***
- $P(x)$ *is the* ***evidence*** *or marginal likelihood*

3.8.3 Bayes Classifier Optimality

Theorem 3.8.2 (Bayes Optimality)**.** *The Bayes classifier minimizes the probability of misclassification among all classifiers.*

Proof Sketch: For a given x, the probability of misclassification is:

$$P(\text{error}|x) = 1 - \max_i P(C_i|x)$$

The Bayes classifier chooses the class with maximum $P(C_i|x)$, thus minimizing $P(\text{error}|x)$ at each x. Integrating over all x gives the minimum overall error rate.

3.8.4 Bayes Decision Boundary

For two classes, the Bayes decision boundary occurs where:

$$P(C_1|x) = P(C_2|x)$$

or equivalently:

$$P(x|C_1)P(C_1) = P(x|C_2)P(C_2)$$

Gaussian Case

When class-conditional densities are Gaussian with equal covariance Σ:

$$P(x|C_i) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T\Sigma^{-1}(x-\mu_i)\right)$$

Taking logs, the decision boundary becomes linear:

$$x^T\Sigma^{-1}(\mu_1 - \mu_2) - \frac{1}{2}(\mu_1^T\Sigma^{-1}\mu_1 - \mu_2^T\Sigma^{-1}\mu_2) + \ln\frac{P(C_1)}{P(C_2)} = 0$$

3.8.5 Multi-Class Classification

For multi-class problems, the Bayes classifier naturally extends by selecting:

$$\hat{C}(x) = \arg \max_{i \in \{1,\ldots,k\}} P(C_i|x)$$

3.9 NAIVE BAYES CLASSIFIER

3.9.1 Class Conditional Independence Assumption

The naive Bayes classifier makes a strong assumption: features are conditionally independent given the class.

Definition 3.9.1 (Naive Bayes Assumption)**.**

$$P(x|C_i) = \prod_{j=1}^{d} P(x_j|C_i)$$

where $x = (x_1, x_2, \ldots, x_d)$ is the feature vector.

3.9.2 Naive Bayes Classifier (NBC)

Definition 3.9.2 (Naive Bayes Classifier)**.** *Using the independence assumption and Bayes' theorem:*

$$P(C_i|x) = \frac{P(C_i) \prod_{j=1}^{d} P(x_j|C_i)}{\sum_{m=1}^{k} P(C_m) \prod_{j=1}^{d} P(x_j|C_m)}$$

The predicted class is:

$$\hat{C}(x) = \arg\max_i P(C_i) \prod_{j=1}^{d} P(x_j|C_i)$$

3.9.3 Estimating Probabilities

Categorical Features

For categorical features:

$$P(x_j = v|C_i) = \frac{\text{count}(x_j = v \text{ and } C_i) + \alpha}{\text{count}(C_i) + \alpha \cdot n_j}$$

where α is the smoothing parameter (often $\alpha = 1$ for Laplace smoothing) and n_j is the number of possible values for feature j.

Numerical Features

For numerical features, common approaches include:

- **Gaussian Naive Bayes**: Assume $P(x_j|C_i) \sim \mathcal{N}(\mu_{ij}, \sigma_{ij}^2)$
- **Discretization**: Convert continuous values into bins

3.9.4 Worked Example: Spam Classification

Consider classifying emails as spam or not spam based on the presence of certain words.

Training Data:

Email	Contains "free"	Contains "money"	Spam
1	Yes	Yes	Yes
2	Yes	No	Yes
3	No	Yes	No
4	No	No	No
5	Yes	Yes	Yes
6	Yes	No	No

Step 1: Calculate Prior Probabilities

$$P(\text{Spam}) = \frac{3}{6} = 0.5$$
$$P(\text{Not Spam}) = \frac{3}{6} = 0.5$$

Step 2: Calculate Conditional Probabilities with Laplace Smoothing ($\alpha = 1$)

For "free" given Spam:

$$P(\text{free} = \text{Yes}|\text{Spam}) = \frac{3+1}{3+2} = \frac{4}{5} = 0.8$$
$$P(\text{free} = \text{No}|\text{Spam}) = \frac{0+1}{3+2} = \frac{1}{5} = 0.2$$

For "free" given Not Spam:

$$P(\text{free} = \text{Yes}|\text{Not Spam}) = \frac{1+1}{3+2} = \frac{2}{5} = 0.4$$
$$P(\text{free} = \text{No}|\text{Not Spam}) = \frac{2+1}{3+2} = \frac{3}{5} = 0.6$$

For "money" given Spam:

$$P(\text{money} = \text{Yes}|\text{Spam}) = \frac{2+1}{3+2} = \frac{3}{5} = 0.6$$
$$P(\text{money} = \text{No}|\text{Spam}) = \frac{1+1}{3+2} = \frac{2}{5} = 0.4$$

For "money" given Not Spam:

$$P(\text{money} = \text{Yes}|\text{Not Spam}) = \frac{1+1}{3+2} = \frac{2}{5} = 0.4$$
$$P(\text{money} = \text{No}|\text{Not Spam}) = \frac{2+1}{3+2} = \frac{3}{5} = 0.6$$

Step 3: Classify a New Email
New email: Contains "free" = Yes, Contains "money" = Yes
Compute posterior for Spam:

$$P(\text{Spam}|\text{free} = \text{Yes}, \text{money} = \text{Yes}) \propto P(\text{Spam}) \times P(\text{free} = \text{Yes}|\text{Spam}) \times P(\text{money} = \text{Yes}|\text{Spam})$$
$$= 0.5 \times 0.8 \times 0.6 = 0.24$$

Compute posterior for Not Spam:

$$P(\text{Not Spam}|\text{free} = \text{Yes}, \text{money} = \text{Yes}) \propto P(\text{Not Spam}) \times P(\text{free} = \text{Yes}|\text{Not Spam}) \times P(\text{money} = \text{Yes}|\text{Not}$$
$$= 0.5 \times 0.4 \times 0.4 = 0.08$$

Normalize:

$$P(\text{Spam}|x) = \frac{0.24}{0.24 + 0.08} = 0.75$$
$$P(\text{Not Spam}|x) = \frac{0.08}{0.24 + 0.08} = 0.25$$

The classifier predicts "Spam" with 75% confidence.

3.9.5 Gaussian Naive Bayes Example

Consider a dataset with continuous features:

Height (cm)	Weight (kg)	Gender
170	65	M
175	70	M
180	75	M
160	55	F
165	58	F
168	60	F

Step 1: Calculate parameters for each class
For Male:

$$\mu_{\text{height}} = \frac{170 + 175 + 180}{3} = 175$$
$$\sigma^2_{\text{height}} = \frac{(170 - 175)^2 + (175 - 175)^2 + (180 - 175)^2}{3} = \frac{25 + 0 + 25}{3} = 16.67$$
$$\mu_{\text{weight}} = \frac{65 + 70 + 75}{3} = 70$$
$$\sigma^2_{\text{weight}} = \frac{(65 - 70)^2 + (70 - 70)^2 + (75 - 70)^2}{3} = \frac{25 + 0 + 25}{3} = 16.67$$

For Female:

$$\mu_{\text{height}} = \frac{160 + 165 + 168}{3} = 164.33$$
$$\sigma^2_{\text{height}} = \frac{(160 - 164.33)^2 + (165 - 164.33)^2 + (168 - 164.33)^2}{3}$$
$$= \frac{18.75 + 0.45 + 13.47}{3} = 10.89$$
$$\mu_{\text{weight}} = \frac{55 + 58 + 60}{3} = 57.67$$
$$\sigma^2_{\text{weight}} = \frac{(55 - 57.67)^2 + (58 - 57.67)^2 + (60 - 57.67)^2}{3}$$
$$= \frac{7.13 + 0.11 + 5.43}{3} = 4.22$$

Step 2: Classify a new person

New person: height = 172 cm, weight = 68 kg

For Male:

$$P(\text{height} = 172|M) = \frac{1}{\sqrt{2\pi \times 16.67}} \exp\left(-\frac{(172 - 175)^2}{2 \times 16.67}\right)$$
$$= \frac{1}{\sqrt{104.7}} \exp\left(-\frac{9}{33.34}\right) = \frac{1}{10.23} \exp(-0.27) = 0.0977 \times 0.763 = 0.0746$$

$$P(\text{weight} = 68|M) = \frac{1}{\sqrt{2\pi \times 16.67}} \exp\left(-\frac{(68 - 70)^2}{2 \times 16.67}\right)$$
$$= \frac{1}{10.23} \exp\left(-\frac{4}{33.34}\right) = 0.0977 \times 0.887 = 0.0867$$

$$P(M) \times P(h|M) \times P(w|M) = 0.5 \times 0.0746 \times 0.0867 = 0.00323$$

For Female:

$$P(\text{height} = 172|F) = \frac{1}{\sqrt{2\pi \times 10.89}} \exp\left(-\frac{(172 - 164.33)^2}{2 \times 10.89}\right)$$
$$= \frac{1}{\sqrt{68.42}} \exp\left(-\frac{58.83}{21.78}\right) = \frac{1}{8.27} \exp(-2.70) = 0.121 \times 0.067 = 0.00811$$

$$P(\text{weight} = 68|F) = \frac{1}{\sqrt{2\pi \times 4.22}} \exp\left(-\frac{(68 - 57.67)^2}{2 \times 4.22}\right)$$
$$= \frac{1}{\sqrt{26.52}} \exp\left(-\frac{106.71}{8.44}\right) = \frac{1}{5.15} \exp(-12.64) = 0.194 \times 3.2 \times 10^{-6} = 6.21 \times 10^{-7}$$

$$P(F) \times P(h|F) \times P(w|F) = 0.5 \times 0.00811 \times 6.21 \times 10^{-7} = 2.52 \times 10^{-9}$$

The classifier predicts "Male" since $0.00323 > 2.52 \times 10^{-9}$.

3.9.6 Advantages and Disadvantages of Naive Bayes

Advantages:

- Simple and fast to train and predict
- Works well with high-dimensional data
- Handles both categorical and continuous features
- Robust to irrelevant features
- Requires small amount of training data

Disadvantages:

- Strong independence assumption rarely holds in real data
- When features are correlated, probabilities can be biased
- Zero probability problem (handled by smoothing)
- Not good for regression tasks

3.10 CASE STUDIES

3.10.1 Case Study 1: Credit Risk Assessment

Problem: A bank wants to automatically assess credit risk for loan applicants.

Dataset: 10,000 applicants with features: income, credit score, employment length, debt-to-income ratio, loan amount, loan purpose, and previous defaults.

Solution: Random Forest Classifier

1. **Data Preprocessing**: Handle missing values, encode categorical variables
2. **Model Building**: Train a random forest with 500 trees, using Gini impurity
3. **Feature Importance**: Found that credit score and debt-to-income ratio were most important
4. **Results**: 92% accuracy, 0.85 F1-score on test set
5. **Interpretation**: The bank can explain to applicants why they were rejected based on the decision paths

3.10.2 Case Study 2: Medical Diagnosis

Problem: Diagnose whether a patient has a particular disease based on symptoms and test results.

Dataset: 5,000 patients with 50 binary features (symptoms present/absent) and disease label.

Solution: Naive Bayes Classifier

1. **Why Naive Bayes?**: Features are binary, independence assumption reasonable for this domain
2. **Model Building**: Train with Laplace smoothing ($\alpha = 1$)
3. **Results**: 88% accuracy, high recall (important for medical diagnosis)
4. **Advantage**: Can provide probability estimates, useful for further clinical decision-making

3.10.3 Case Study 3: Customer Churn Prediction

Problem: Telecom company wants to predict which customers are likely to churn.

Dataset: 100,000 customers with 20 features: usage patterns, contract type, customer service calls, tenure, monthly charges.

Solution: Decision Tree with Pruning

1. **Model Building**: Build full tree then prune using cross-validation
2. **Resulting Tree**: Simple tree with 8 leaves, easy to interpret
3. **Key Insights**:
   - Customers with tenure < 6 months and high monthly charges are most likely to churn
   - Customers with fiber optic service and frequent customer service calls have high churn risk
4. **Business Impact**: Company can target high-risk customers with retention offers

3.11 EXERCISES

3.11.1 Theoretical Exercises

1. Prove that entropy is maximized when classes are equally distributed.
2. Derive the relationship between Gini index and entropy for binary classification.
3. Show that the Bayes classifier is optimal in terms of minimizing misclassification rate.

4. Prove that the bias-variance decomposition holds for squared error loss.

5. Explain why random forests do not require a separate validation set for error estimation.

3.11.2 Practical Exercises

1. Build a decision tree for the Iris dataset using information gain. Show all calculations.

2. Implement Naive Bayes classifier for the Play Tennis dataset and classify a new instance: (Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong).

3. For the regression tree example, calculate the reduction in variance achieved by splitting at size=1600.

4. Given a confusion matrix, calculate precision, recall, and F1-score for a binary classifier.

5. Perform bias-variance decomposition for a decision tree of varying depths on a synthetic dataset.

3.12 SOLUTIONS TO SELECTED EXERCISES

3.12.1 Solution to Exercise 1.1

Problem: Prove that entropy is maximized when classes are equally distributed.

Proof: For k classes, entropy is $H = -\sum_{i=1}^{k} p_i \log p_i$ with constraint $\sum_{i=1}^{k} p_i = 1$.

Using Lagrange multipliers:

$$L = -\sum_{i=1}^{k} p_i \log p_i + \lambda \left(\sum_{i=1}^{k} p_i - 1 \right)$$

Taking derivative with respect to p_i:

$$\frac{\partial L}{\partial p_i} = -\log p_i - 1 + \lambda = 0$$

This gives $\log p_i = \lambda - 1$, so all p_i are equal: $p_i = \frac{1}{k}$ for all i.

Thus, entropy is maximized when classes are equally distributed, with maximum value $H_{\max} = \log k$.

3.12.2 Solution to Exercise 2.2

Problem: Classify using Naive Bayes: (Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

From the Play Tennis dataset:

$$P(\text{Yes}) = 9/14 = 0.643$$
$$P(\text{No}) = 5/14 = 0.357$$

Conditional probabilities with Laplace smoothing:

$$\begin{aligned}
P(\text{Sunny}|\text{Yes}) &= (2+1)/(9+3) = 3/12 = 0.25 \\
P(\text{Sunny}|\text{No}) &= (3+1)/(5+3) = 4/8 = 0.5 \\
P(\text{Cool}|\text{Yes}) &= (3+1)/(9+3) = 4/12 = 0.333 \\
P(\text{Cool}|\text{No}) &= (1+1)/(5+3) = 2/8 = 0.25 \\
P(\text{High}|\text{Yes}) &= (3+1)/(9+3) = 4/12 = 0.333 \\
P(\text{High}|\text{No}) &= (4+1)/(5+3) = 5/8 = 0.625 \\
P(\text{Strong}|\text{Yes}) &= (3+1)/(9+3) = 4/12 = 0.333 \\
P(\text{Strong}|\text{No}) &= (3+1)/(5+3) = 4/8 = 0.5
\end{aligned}$$

Posterior for Yes:

$$\begin{aligned}
P(\text{Yes}|x) &\propto 0.643 \times 0.25 \times 0.333 \times 0.333 \times 0.333 \\
&= 0.643 \times 0.25 \times 0.037 = 0.00595
\end{aligned}$$

Posterior for No:

$$\begin{aligned}
P(\text{No}|x) &\propto 0.357 \times 0.5 \times 0.25 \times 0.625 \times 0.5 \\
&= 0.357 \times 0.5 \times 0.25 \times 0.625 \times 0.5 = 0.0139
\end{aligned}$$

Normalizing:

$$\begin{aligned}
P(\text{Yes}|x) &= \frac{0.00595}{0.00595 + 0.0139} = 0.30 \\
P(\text{No}|x) &= \frac{0.0139}{0.01985} = 0.70
\end{aligned}$$

The classifier predicts "No" (Don't Play) with 70% probability.

Chapter 4

Linear Discriminants for Machine Learning

4.1 Introduction to Linear Discriminants

4.1.1 Mathematical Framework

Linear discriminants are functions that partition the feature space into decision regions using linear boundaries. A linear discriminant function is defined as:

$$g(x) = w^T x + b = \sum_{i=1}^{d} w_i x_i + b$$

where:

- $x \in \mathbb{R}^d$: Input feature vector
- $w \in \mathbb{R}^d$: Weight vector (normal to decision boundary)
- $b \in \mathbb{R}$: Bias term (offset from origin)

The decision rule for binary classification:

$$\hat{y} = \begin{cases} +1 & \text{if } g(x) > 0 \\ -1 & \text{if } g(x) < 0 \\ \text{undecided} & \text{if } g(x) = 0 \end{cases}$$

4.1.2 Geometric Interpretation

The decision boundary is a hyperplane:

$$\{x : w^T x + b = 0\}$$

Properties:

- w is normal to the hyperplane

- Distance from a point x to the hyperplane: $\frac{|w^T x+b|}{\|w\|}$
- Signed distance: $\frac{w^T x+b}{\|w\|}$
- The bias term b determines the offset from origin

4.2 Linear Discriminants for Classification

4.2.1 Fisher's Linear Discriminant

Fisher's criterion maximizes class separation by projecting data onto a line:

$$J(w) = \frac{(\mu_1 - \mu_2)^2}{s_1^2 + s_2^2} = \frac{w^T S_B w}{w^T S_W w}$$

where:

$$\mu_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \quad \text{(class means)}$$
$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \quad \text{(between-class scatter)}$$
$$S_W = \sum_{k=1}^{2} \sum_{i:y_i=k} (x_i - \mu_k)(x_i - \mu_k)^T \quad \text{(within-class scatter)}$$

The optimal solution:

$$w \propto S_W^{-1}(\mu_1 - \mu_2)$$

4.2.2 Mathematical Derivation

Maximizing $J(w)$ leads to the generalized eigenvalue problem:

$$S_B w = \lambda S_W w$$

Since $S_B w$ is always in the direction of $(\mu_1 - \mu_2)$, we get:

$$w \propto S_W^{-1}(\mu_1 - \mu_2)$$

4.2.3 Example Problem 4.1: Fisher's Linear Discriminant

Problem: Given two classes in 2D: Class 1: $\{(2,3),(3,4),(2,4)\}$ Class 2: $\{(5,7),(6,8),(5,8)\}$ Find Fisher's discriminant direction.

Solution:

Class 1 mean:

$$\mu_1 = \left(\frac{2+3+2}{3}, \frac{3+4+4}{3}\right) = \left(\frac{7}{3}, \frac{11}{3}\right) \approx (2.33, 3.67)$$

Class 2 mean:

$$\mu_2 = \left(\frac{5+6+5}{3}, \frac{7+8+8}{3}\right) = \left(\frac{16}{3}, \frac{23}{3}\right) \approx (5.33, 7.67)$$

Difference in means:

$$\mu_1 - \mu_2 = (-3, -4)$$

Within-class scatter matrix calculation:
For Class 1:

$$S_1 = \sum_{i \in C_1} (x_i - \mu_1)(x_i - \mu_1)^T$$

$$= \begin{bmatrix} (2-2.33)^2 & (2-2.33)(3-3.67) \\ (3-3.67)(2-2.33) & (3-3.67)^2 \end{bmatrix} + \begin{bmatrix} (3-2.33)^2 & (3-2.33)(4-3.67) \\ (4-3.67)(3-2.33) & (4-3.67)^2 \end{bmatrix} + \begin{bmatrix} (2-2 \\ (4-3.67)(\end{bmatrix}$$

$$S_1 \approx \begin{bmatrix} 0.89 & 0.44 \\ 0.44 & 0.89 \end{bmatrix}$$

For Class 2:

$$S_2 \approx \begin{bmatrix} 0.89 & 0.44 \\ 0.44 & 0.89 \end{bmatrix}$$

Total within-class scatter:

$$S_W = S_1 + S_2 = \begin{bmatrix} 1.78 & 0.88 \\ 0.88 & 1.78 \end{bmatrix}$$

Inverse of S_W:

$$S_W^{-1} = \frac{1}{(1.78)^2 - (0.88)^2} \begin{bmatrix} 1.78 & -0.88 \\ -0.88 & 1.78 \end{bmatrix} \approx \begin{bmatrix} 0.71 & -0.35 \\ -0.35 & 0.71 \end{bmatrix}$$

Fisher's discriminant direction:

$$w \propto S_W^{-1}(\mu_1 - \mu_2) = \begin{bmatrix} 0.71 & -0.35 \\ -0.35 & 0.71 \end{bmatrix} \begin{bmatrix} -3 \\ -4 \end{bmatrix} = \begin{bmatrix} -0.73 \\ -1.79 \end{bmatrix}$$

Thus, the optimal projection vector is approximately $w = (-0.73, -1.79)$ or any scalar multiple thereof.

4.3 Perceptron Classifier

4.3.1 The Perceptron Model

The perceptron is the simplest neural network model, consisting of a single neuron with a step activation function:

$$\hat{y} = \text{sign}(w^T x + b) = \begin{cases} +1 & \text{if } w^T x + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

4.3.2 Geometric Interpretation

The perceptron learns a hyperplane that separates the classes:

- Points with $w^T x + b > 0$ are classified as positive
- Points with $w^T x + b < 0$ are classified as negative
- The weight vector w is perpendicular to the decision boundary

4.4 Perceptron Learning Algorithm

4.4.1 Algorithm Description

The perceptron learning algorithm updates weights when misclassifications occur:

Algorithm 3 Perceptron Learning Algorithm

Require: Training data $\{(x_i, y_i)\}_{i=1}^n$, learning rate $\eta > 0$
Ensure: Learned weights w, bias b

Initialize $w \leftarrow 0$, $b \leftarrow 0$
$t \leftarrow 0$
repeat
    $m \leftarrow 0$ ▷ Count misclassifications
    for $i = 1$ to n **do**
        $\hat{y}_i \leftarrow \text{sign}(w^T x_i + b)$
        if $\hat{y}_i \neq y_i$ **then**
            $w \leftarrow w + \eta y_i x_i$
            $b \leftarrow b + \eta y_i$
            $m \leftarrow m + 1$
        end if
    end for
    $t \leftarrow t + 1$
until $m = 0$ or $t \geq t_{\max}$
return w, b

4.4.2 Convergence Theorem

Theorem 4.4.1 (Perceptron Convergence Theorem)**.** *If the training data is linearly separable, the perceptron algorithm converges to a separating hyperplane in a finite number of steps.*

Proof Sketch: Let w^* be a unit vector that separates the data with margin $\gamma > 0$:

$$y_i(w^{*T} x_i + b^*) \geq \gamma \quad \forall i$$

Define $W_t = (w_t, b_t)$. The update rule ensures:

$$\|W_{t+1} - \alpha W^*\|^2 \leq \|W_t - \alpha W^*\|^2 - 2\alpha\gamma + \eta^2 R^2$$

where $R = \max_i \|x_i\|$. Choosing appropriate α and η guarantees convergence.

4.4.3 Example Problem 4.2: Perceptron Learning

Problem: Train a perceptron on data: Positive: $(1,2), (2,1)$ Negative: $(4,3), (5,4)$ Use $\eta = 1$, initialize $w = (0,0)$, $b = 0$.

Solution:

Epoch 1:

$$\begin{aligned}
\text{Point } (1,2): &\quad w^T x + b = 0 \cdot 1 + 0 \cdot 2 + 0 = 0 \to \hat{y} = 1 \text{ (correct)} \\
\text{Point } (2,1): &\quad 0 \to \hat{y} = 1 \text{ (correct)} \\
\text{Point } (4,3): &\quad 0 \to \hat{y} = 1 \text{ (incorrect, should be -1)} \\
&\quad w \leftarrow (0,0) + 1 \cdot (-1) \cdot (4,3) = (-4,-3) \\
&\quad b \leftarrow 0 + 1 \cdot (-1) = -1 \\
\text{Point } (5,4): &\quad (-4,-3) \cdot (5,4) - 1 = -20 - 12 - 1 = -33 \to \hat{y} = -1 \text{ (correct)}
\end{aligned}$$

Epoch 2:

$$\begin{aligned}
(1,2): &\quad (-4,-3) \cdot (1,2) - 1 = -4 - 6 - 1 = -11 \to \hat{y} = -1 \text{ (incorrect)} \\
&\quad w \leftarrow (-4,-3) + 1 \cdot (+1) \cdot (1,2) = (-3,-1) \\
&\quad b \leftarrow -1 + 1 \cdot (+1) = 0 \\
(2,1): &\quad (-3,-1) \cdot (2,1) + 0 = -6 - 1 = -7 \to \hat{y} = -1 \text{ (incorrect)} \\
&\quad w \leftarrow (-3,-1) + 1 \cdot (+1) \cdot (2,1) = (-1,0) \\
&\quad b \leftarrow 0 + 1 = 1 \\
(4,3): &\quad (-1,0) \cdot (4,3) + 1 = -4 + 1 = -3 \to \hat{y} = -1 \text{ (correct)} \\
(5,4): &\quad (-1,0) \cdot (5,4) + 1 = -5 + 1 = -4 \to \hat{y} = -1 \text{ (correct)}
\end{aligned}$$

Continue until convergence. The algorithm finds a separating hyperplane.

4.5 Support Vector Machines (SVM)

4.5.1 Mathematical Formulation

SVM finds the hyperplane that maximizes the margin between classes. For linearly separable data:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 \quad \forall i$$

4.5.2 Geometric Interpretation

- Margin: $\frac{2}{\|w\|}$
- Support vectors: Points with $y_i(w^T x_i + b) = 1$
- Decision boundary: $w^T x + b = 0$

4.5.3 Optimization Problem

The primal Lagrangian:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1]$$

where $\alpha_i \geq 0$ are Lagrange multipliers.

4.5.4 Dual Formulation

Taking derivatives and substituting back:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i$$

The decision function becomes:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (x_i^T x) + b \right)$$

where only points with $\alpha_i > 0$ (support vectors) contribute.

4.5.5 Example Problem 4.3: SVM Computation

Problem: Find SVM for points: Positive: $(2, 2)$, Negative: $(0, 0)$

Solution: The maximum margin hyperplane is perpendicular to the line connecting the points.

$$w = (2, 2) - (0, 0) = (2, 2)$$
$$\|w\| = \sqrt{4 + 4} = \sqrt{8} \approx 2.828$$
$$\text{Margin} = \frac{2}{\|w\|} = \frac{2}{\sqrt{8}} = \frac{1}{\sqrt{2}} \approx 0.707$$

The optimal hyperplane is the perpendicular bisector:

$$(2, 2) \cdot (x - (1, 1)) = 0 \Rightarrow 2x_1 + 2x_2 - 4 = 0 \Rightarrow x_1 + x_2 - 2 = 0$$

4.6 Linearly Non-Separable Case

4.6.1 Soft Margin SVM

Introduce slack variables $\xi_i \geq 0$ to allow misclassifications:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad \forall i$$

4.6.2 Dual Formulation with Slack

The dual problem becomes:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

subject to:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C \quad \forall i$$

4.6.3 Interpretation of C

- C large: Hard margin, few support vectors, may overfit
- C small: Soft margin, more support vectors, may underfit
- Tradeoff between margin width and misclassification

4.7 Non-linear SVM and Kernel Trick

4.7.1 Motivation

For non-linearly separable data, map to higher-dimensional feature space:

$$\phi : \mathbb{R}^d \to \mathcal{F}$$

4.7.2 Kernel Trick

The dual formulation only requires dot products:

$$x_i^T x_j \to \phi(x_i)^T \phi(x_j) = K(x_i, x_j)$$

where K is a kernel function.

Table 4.1: Kernel Functions for SVM

Kernel Type	**Mathematical Form**	**Parameters**
Linear	$K(x, y) = x^T y$	-
Polynomial	$K(x, y) = (x^T y + c)^d$	$c \geq 0$, $d \in \mathbb{N}$
RBF/Gaussian	$K(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$	$\sigma > 0$
Sigmoid	$K(x, y) = \tanh(\kappa x^T y + \theta)$	$\kappa > 0$, $\theta < 0$
Laplacian	$K(x, y) = \exp\left(-\frac{\|x-y\|}{\sigma}\right)$	$\sigma > 0$

4.7.3 Common Kernel Functions

4.7.4 Mercer's Theorem

Theorem 4.7.1 (Mercer's Theorem)**.** *A symmetric function $K(x, y)$ is a valid kernel iff for any finite set of points $\{x_1, \ldots, x_n\}$, the Gram matrix $K_{ij} = K(x_i, x_j)$ is positive semi-definite.*

4.7.5 Example Problem 4.4: Kernel SVM

Problem: Given points in 1D: Positive: $\{-2, 2\}$, Negative: $\{0\}$. Show that using polynomial kernel of degree 2 makes them linearly separable.

Solution: Map with $\phi(x) = (x, x^2)$:

$$x = -2 \to \phi = (-2, 4)$$
$$x = 0 \to \phi = (0, 0)$$
$$x = 2 \to \phi = (2, 4)$$

In this 2D space, points $(-2, 4)$ and $(2, 4)$ are separable from $(0, 0)$ by line $x_2 = 1$.

4.8 Logistic Regression

4.8.1 Mathematical Model

Logistic regression models probability using the sigmoid function:

$$P(y = 1|x) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the logistic sigmoid.

4.8.2 Likelihood and Loss Function

For binary classification with $y \in \{0, 1\}$, the likelihood:

$$L(w, b) = \prod_{i=1}^{n} P(y_i|x_i)^{y_i}(1 - P(y_i|x_i))^{1-y_i}$$

Negative log-likelihood (cross-entropy loss):

$$\mathcal{L}(w, b) = -\sum_{i=1}^{n} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

where $\hat{y}_i = \sigma(w^T x_i + b)$.

4.8.3 Gradient Derivation

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{i=1}^{n} (\hat{y}_i - y_i) x_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} (\hat{y}_i - y_i)$$

4.8.4 Example Problem 4.5: Logistic Regression

Problem: For data point $x = 2$, $y = 1$, with current parameters $w = 0.5$, $b = 0$, learning rate $\eta = 0.1$, perform one gradient update.

Solution:

$$z = 0.5 \times 2 + 0 = 1$$

$$\hat{y} = \sigma(1) = \frac{1}{1 + e^{-1}} \approx 0.731$$

$$\frac{\partial \mathcal{L}}{\partial w} = (\hat{y} - y)x = (0.731 - 1) \times 2 = -0.538$$

$$\frac{\partial \mathcal{L}}{\partial b} = \hat{y} - y = 0.731 - 1 = -0.269$$

$$w_{\text{new}} = w - \eta \frac{\partial \mathcal{L}}{\partial w} = 0.5 - 0.1 \times (-0.538) = 0.554$$

$$b_{\text{new}} = b - \eta \frac{\partial \mathcal{L}}{\partial b} = 0 - 0.1 \times (-0.269) = 0.027$$

4.9 Linear Regression (Review)

4.9.1 Matrix Formulation

Linear regression models the target as:

$$y = X\beta + \epsilon$$

where $X \in \mathbb{R}^{n \times d}$, $\beta \in \mathbb{R}^d$, $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$.

4.9.2 Ordinary Least Squares Solution

Minimize MSE: $\min_\beta \|y - X\beta\|_2^2$

Solution: $\hat{\beta} = (X^T X)^{-1} X^T y$

4.9.3 Connection to Logistic Regression

- Linear regression: Models continuous output, Gaussian likelihood
- Logistic regression: Models binary output, Bernoulli likelihood
- Both are Generalized Linear Models (GLMs)

4.10 Multi-Layer Perceptrons (MLPs)

4.10.1 Architecture

An MLP consists of layers of neurons:

$$h^{(l)} = \sigma^{(l)}(W^{(l)}h^{(l-1)} + b^{(l)})$$

where:

- $h^{(0)} = x$ (input layer)
- $h^{(L)}$ (output layer)
- $W^{(l)}$: Weight matrix for layer l
- $b^{(l)}$: Bias vector for layer l
- $\sigma^{(l)}$: Activation function

4.10.2 Activation Functions

Table 4.2: Common Activation Functions

Function	**Formula**	**Properties**
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$	Output in $(0, 1)$, vanishing gradient
Tanh	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Output in $(-1, 1)$, zero-centered
ReLU	$\text{ReLU}(z) = \max(0, z)$	Sparse activation, no vanishing gradient
Leaky ReLU	$\text{LReLU}(z) = \max(\alpha z, z)$	Avoids dead neurons
Softmax	$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$	Probability distribution for classification

4.10.3 Universal Approximation Theorem

Theorem 4.10.1 (Cybenko, 1989)**.** *A feedforward network with a single hidden layer containing a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.*

4.11 Backpropagation for Training an MLP

4.11.1 Forward Propagation

For a network with L layers:

$$\begin{aligned} a^{(1)} &= W^{(1)}x + b^{(1)} \\ h^{(1)} &= \sigma^{(1)}(a^{(1)}) \\ a^{(2)} &= W^{(2)}h^{(1)} + b^{(2)} \\ &\vdots \\ \hat{y} = a^{(L)} &= W^{(L)}h^{(L-1)} + b^{(L)} \end{aligned}$$

4.11.2 Loss Function

For regression with MSE:

$$\mathcal{L} = \frac{1}{2}\|y - \hat{y}\|_2^2$$

For classification with cross-entropy:

$$\mathcal{L} = -\sum_{j=1}^{K} y_j \log \hat{y}_j$$

4.11.3 Backpropagation Algorithm

Using chain rule, compute gradients backward:

For output layer ($l = L$):

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial a^{(L)}} = \hat{y} - y \quad \text{(for MSE)}$$

For hidden layers ($l = L-1, \ldots, 1$):

$$\delta^{(l)} = (\sigma^{(l)})'(a^{(l)}) \odot (W^{(l+1)})^T \delta^{(l+1)}$$

where $\odot$ denotes element-wise multiplication.

Gradients:

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)}(h^{(l-1)})^T$$

$$\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{(l)}$$

4.11.4 Example Problem 4.6: Backpropagation

Problem: Consider a 2-layer network with:

- Input: $x = [1, 1]^T$
- Hidden layer: $W^{(1)} = \begin{bmatrix} 0.5 & 0.3 \\ 0.2 & 0.4 \end{bmatrix}$, $b^{(1)} = [0.1, 0.2]^T$, $\sigma = \text{sigmoid}$
- Output layer: $W^{(2)} = [0.6, 0.7]$, $b^{(2)} = 0.3$, linear output
- Target: $y = 0.8$, learning rate $\eta = 0.1$

Perform one forward and backward pass.

Solution:

Forward Pass:

Hidden layer pre-activation:

$$a^{(1)} = W^{(1)}x + b^{(1)} = \begin{bmatrix} 0.5 & 0.3 \\ 0.2 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

$$a^{(1)} = \begin{bmatrix} 0.5 + 0.3 \\ 0.2 + 0.4 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.8 \end{bmatrix}$$

Hidden layer activation (sigmoid):

$$h^{(1)} = \sigma(a^{(1)}) = \begin{bmatrix} \sigma(0.9) \\ \sigma(0.8) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-0.9}} \\ \frac{1}{1+e^{-0.8}} \end{bmatrix} \approx \begin{bmatrix} 0.711 \\ 0.690 \end{bmatrix}$$

Output layer pre-activation:

$$a^{(2)} = W^{(2)}h^{(1)} + b^{(2)} = [0.6, 0.7] \begin{bmatrix} 0.711 \\ 0.690 \end{bmatrix} + 0.3$$

$$a^{(2)} = 0.6 \times 0.711 + 0.7 \times 0.690 + 0.3 = 0.427 + 0.483 + 0.3 = 1.210$$

Final output (linear activation):

$$\hat{y} = a^{(2)} = 1.210$$

Loss (MSE):

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(0.8 - 1.210)^2 = \frac{1}{2}(-0.410)^2 = 0.5 \times 0.1681 = 0.08405$$

Backward Pass:

Output layer error:

$$\delta^{(2)} = \frac{\partial \mathcal{L}}{\partial a^{(2)}} = \hat{y} - y = 1.210 - 0.8 = 0.410$$

Output layer gradients:

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}} = \delta^{(2)}(h^{(1)})^T = 0.410 \times [0.711, 0.690] = [0.292, 0.283]$$

$$\frac{\partial \mathcal{L}}{\partial b^{(2)}} = \delta^{(2)} = 0.410$$

Hidden layer derivative (sigmoid):

$$\sigma'(a^{(1)}) = \sigma(a^{(1)}) \odot (1 - \sigma(a^{(1)})) = \begin{bmatrix} 0.711 \times (1 - 0.711) \\ 0.690 \times (1 - 0.690) \end{bmatrix} = \begin{bmatrix} 0.206 \\ 0.214 \end{bmatrix}$$

Hidden layer error:

$$\delta^{(1)} = \sigma'(a^{(1)}) \odot \left((W^{(2)})^T \delta^{(2)}\right) = \begin{bmatrix} 0.206 \\ 0.214 \end{bmatrix} \odot \left(\begin{bmatrix} 0.6 \\ 0.7 \end{bmatrix} \times 0.410 \right)$$

$$\delta^{(1)} = \begin{bmatrix} 0.206 \\ 0.214 \end{bmatrix} \odot \begin{bmatrix} 0.246 \\ 0.287 \end{bmatrix} = \begin{bmatrix} 0.051 \\ 0.061 \end{bmatrix}$$

Hidden layer gradients:

$$\frac{\partial \mathcal{L}}{\partial W^{(1)}} = \delta^{(1)} x^T = \begin{bmatrix} 0.051 \\ 0.061 \end{bmatrix} [1, 1] = \begin{bmatrix} 0.051 & 0.051 \\ 0.061 & 0.061 \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(1)}} = \delta^{(1)} = \begin{bmatrix} 0.051 \\ 0.061 \end{bmatrix}$$

Parameter Updates:
Output layer:

$$W^{(2)}_{\text{new}} = W^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(2)}} = [0.6, 0.7] - 0.1 \times [0.292, 0.283] = [0.571, 0.672]$$

$$b^{(2)}_{\text{new}} = b^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(2)}} = 0.3 - 0.1 \times 0.410 = 0.259$$

Hidden layer:

$$W^{(1)}_{\text{new}} = W^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(1)}} = \begin{bmatrix} 0.5 & 0.3 \\ 0.2 & 0.4 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0.051 & 0.051 \\ 0.061 & 0.061 \end{bmatrix} = \begin{bmatrix} 0.495 & 0.295 \\ 0.194 & 0.394 \end{bmatrix}$$

$$b^{(1)}_{\text{new}} = b^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(1)}} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix} - 0.1 \times \begin{bmatrix} 0.051 \\ 0.061 \end{bmatrix} = \begin{bmatrix} 0.095 \\ 0.194 \end{bmatrix}$$

4.11.5 Optimization Algorithms

Stochastic Gradient Descent (SGD)

Update after each example:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_i(\theta)$$

Mini-batch SGD

Update after small batch of m examples:

$$\theta \leftarrow \theta - \frac{\eta}{m} \sum_{i=1}^{m} \nabla_\theta \mathcal{L}_i(\theta)$$

Momentum

Accelerates convergence:

$$v_t = \beta v_{t-1} + \nabla_\theta \mathcal{L}(\theta)$$
$$\theta \leftarrow \theta - \eta v_t$$

Adam (Adaptive Moment Estimation)

Combines momentum with adaptive learning rates:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta))^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ \theta &\leftarrow \theta - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}$$

4.12 Regularization in Neural Networks

4.12.1 L1 and L2 Regularization

Add penalty to loss:

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$$

4.12.2 Dropout

Randomly drop neurons during training:

$$h_{\text{drop}} = h \odot \text{Bernoulli}(p)$$

where p is the keep probability.

4.12.3 Batch Normalization

Normalize layer inputs:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$
$$y = \gamma \hat{x} + \beta$$

4.13 Early Stopping

Monitor validation error and stop when it starts increasing:

$$\text{Stop when } \text{Err}_{\text{val}}(t) > \text{Err}_{\text{val}}(t-p) \text{ for } p \text{ consecutive epochs}$$

4.14 Multiple Choice Questions

4.14.1 Knowledge Level Questions

1. The Perceptron convergence theorem guarantees convergence if:
   (a) Data is linearly separable (Correct)
   (b) Learning rate is small
   (c) Data is normalized
   (d) Network has multiple layers
2. The kernel trick allows SVMs to:
   (a) Reduce computational complexity
   (b) Handle non-linear decision boundaries (Correct)
   (c) Work with unlabeled data
   (d) Automatically tune hyperparameters
3. In soft-margin SVM, the parameter C controls:
   (a) **Tradeoff between margin width and misclassification** (Correct)
   (b) Learning rate
   (c) Kernel complexity
   (d) Number of support vectors

4.14.2 Comprehension Level Questions

4. The vanishing gradient problem in deep networks refers to:
   (a) Gradients becoming zero in early layers (Correct)
   (b) Loss function approaching zero
   (c) Weights becoming too large
   (d) Training accuracy reaching 100%
5. ReLU activation helps with vanishing gradients because:
   (a) It is symmetric
   (b) **Its derivative is 1 for positive inputs** (Correct)
   (c) It outputs values between 0 and 1
   (d) It is differentiable everywhere

4.14.3 Application Level Questions

6. For logistic regression, if $w^T x + b = 2$, then $P(y = 1|x)$ is approximately:

   (a) 0.12

   (b) 0.27

   (c) **0.88** (Correct)

   (d) 0.98

   Calculation: $\sigma(2) = 1/(1 + e^{-2}) \approx 0.88$

7. Given kernel $K(x, y) = (x^T y + 1)^3$, this corresponds to a polynomial kernel of degree:

   (a) 1

   (b) 2

   (c) **3** (Correct)

   (d) 4

4.14.4 Analysis Level Questions

8. In backpropagation, the error term $\delta^{(l)}$ for a hidden layer depends on:

   (a) Only the next layer's error term (Correct)

   (b) Only the previous layer's error term

   (c) Both next and previous layer errors

   (d) Only the output error

9. If training error is low but validation error is high, this indicates:

   (a) Underfitting

   (b) **Overfitting** (Correct)

   (c) Optimal fit

   (d) Need more training data

4.14.5 Synthesis Level Questions

10. For a non-linearly separable problem, which approach is most appropriate?

    (a) Linear SVM

    (b) Perceptron

    (c) **Kernel SVM or MLP** (Correct)

    (d) Linear regression

11. When using ReLU activation, some neurons become permanently inactive (die). This can be addressed by:

    (a) Using sigmoid activation

    (b) **Using Leaky ReLU** (Correct)

    (c) Increasing learning rate

    (d) Decreasing network depth

4.15 Essay Questions and Solutions

1. **Question**: Derive the dual formulation of the SVM optimization problem and explain the significance of support vectors.

   Solution: Starting from primal:

   $$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 \quad \forall i$$

   Lagrangian:

   $$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1]$$

   KKT conditions:

   $$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

   $$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

   Substituting back:

   $$\mathcal{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

   Support Vectors: Points with $\alpha_i > 0$ lie on the margin boundaries ($y_i(w^T x_i + b) = 1$) and completely determine the solution.

2. **Question**: Explain the backpropagation algorithm with mathematical derivations and discuss how it enables learning in multi-layer neural networks.

   Solution: Backpropagation computes gradients using chain rule:

   For output layer:

   $$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(L)}} = \frac{\partial \mathcal{L}}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial w_{jk}^{(L)}} = \delta_j^{(L)} h_k^{(L-1)}$$

For hidden layers:

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} = \sum_m \frac{\partial \mathcal{L}}{\partial a_m^{(l+1)}} \cdot \frac{\partial a_m^{(l+1)}}{\partial a_j^{(l)}} = \sum_m \delta_m^{(l+1)} w_{mj}^{(l+1)} (\sigma)'(a_j^{(l)})$$

Learning Process:

(a) Forward pass: Compute activations

(b) Compute loss

(c) Backward pass: Compute error gradients

(d) Update weights: $w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$

3. **Question**: Compare and contrast Logistic Regression, SVM, and Neural Networks for classification tasks, discussing their mathematical formulations, strengths, and limitations.

   Solution:

   Logistic Regression:

   $$P(y = 1|x) = \sigma(w^T x + b)$$

   - Strengths: Probabilistic output, convex optimization
   - Limitations: Linear decision boundary

   SVM:

   $$f(x) = \text{sign}\left(\sum_i \alpha_i y_i K(x_i, x) + b\right)$$

   - Strengths: Maximum margin, kernel trick
   - Limitations: Not probabilistic, kernel selection

   Neural Networks:

   $$h^{(l)} = \sigma(W^{(l)} h^{(l-1)} + b^{(l)})$$

   - Strengths: Universal approximation, feature learning
   - Limitations: Non-convex, requires large data

4.16 Summary and Key Takeaways

4.16.1 Mathematical Foundations

- **Linear discriminant**: $g(x) = w^T x + b$
- **Fisher's discriminant**: $w \propto S_W^{-1}(\mu_1 - \mu_2)$
- **Perceptron update**: $w \leftarrow w + \eta y_i x_i$ for misclassifications

- **SVM dual**: $\max_\alpha \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(x_i, x_j)$
- **Logistic regression**: $P(y = 1|x) = \sigma(w^T x + b)$
- **Backpropagation**: $\delta^{(l)} = (\sigma)'(a^{(l)}) \odot (W^{(l+1)})^T \delta^{(l+1)}$

4.16.2 Key Insights

1. **Linearity vs Non-linearity**: Linear models work well for simple problems; kernels and neural networks handle complex patterns
2. **Margin maximization**: SVM's max-margin principle improves generalization
3. **Gradient-based learning**: Backpropagation enables training of deep networks
4. **Regularization**: Essential for preventing overfitting in complex models
5. **Activation functions**: Choice affects gradient flow and learning dynamics

4.16.3 Important Formulas

Sigmoid: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$

Cross-entropy loss: $\mathcal{L} = -\sum [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$

SVM objective: $\min \dfrac{1}{2} \|w\|^2 + C \sum \xi_i$

Kernel trick: $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$

ReLU: $\text{ReLU}(z) = \max(0, z)$

Softmax: $\sigma(z)_j = \dfrac{e^{z_j}}{\sum_k e^{z_k}}$

4.16.4 Practical Guidelines

1. Start with simple linear models before trying complex ones
2. Use cross-validation for hyperparameter tuning
3. Monitor overfitting with validation curves
4. Choose activation functions based on problem type
5. Regularize to control model complexity
6. Use appropriate loss functions for the task

Note: Linear discriminants and neural networks form the backbone of modern machine learning. Understanding their mathematical foundations is crucial for effective application and development of new methods.

Chapter 5

Clustering

5.1 Introduction to Clustering

5.1.1 Mathematical Framework

Clustering is an unsupervised learning task that partitions data into groups (clusters) such that objects within the same cluster are more similar to each other than to objects in different clusters.

Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$ with $x_i \in \mathbb{R}^d$, clustering aims to find a partition:

$$\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$$

such that:

$$\bigcup_{j=1}^{k} C_j = X, \quad C_i \cap C_j = \emptyset \text{ for } i \neq j \text{ (for hard clustering)}$$

5.1.2 Objective Functions

The quality of clustering is measured by:

$$J = \sum_{j=1}^{k} \sum_{x_i \in C_j} d(x_i, \mu_j)^2$$

where μ_j is the centroid of cluster C_j and d is a distance measure (typically Euclidean).

5.2 Partitioning of Data

5.2.1 Data Representation

Data can be represented in two forms:

Data Matrix

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

Dissimilarity Matrix

$$D = \begin{bmatrix} 0 & d_{12} & \cdots & d_{1n} \\ d_{21} & 0 & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & 0 \end{bmatrix}$$

where $d_{ij} = d(x_i, x_j)$ is the distance between points i and j.

5.3 Matrix Factorization for Clustering

5.3.1 Spectral Decomposition

For a symmetric matrix A, spectral decomposition:

$$A = U\Lambda U^T = \sum_{i=1}^{n} \lambda_i u_i u_i^T$$

where U contains eigenvectors and Λ contains eigenvalues.

5.3.2 Non-negative Matrix Factorization (NMF)

Given $X \in \mathbb{R}_{\geq 0}^{n \times d}$, find:

$$X \approx WH$$

where $W \in \mathbb{R}_{\geq 0}^{n \times k}$, $H \in \mathbb{R}_{\geq 0}^{k \times d}$, and k is the number of clusters.

Objective function:

$$\min_{W,H \geq 0} \|X - WH\|_F^2$$

5.3.3 Example Problem 5.1: Matrix Factorization

Problem: Given data matrix $X = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 4 & 3 \\ 3 & 4 \end{bmatrix}$, perform NMF with $k = 2$.

Solution: Using multiplicative update rules:

$$H \leftarrow H \odot \frac{W^T X}{W^T W H}, \quad W \leftarrow W \odot \frac{X H^T}{W H H^T}$$

After convergence (simplified):

$$W \approx \begin{bmatrix} 1.2 & 0.8 \\ 1.1 & 0.9 \\ 0.9 & 1.1 \\ 0.8 & 1.2 \end{bmatrix}, \quad H \approx \begin{bmatrix} 0.9 & 1.1 \\ 1.1 & 0.9 \end{bmatrix}$$

The rows of W indicate cluster memberships: points 1,2 belong to cluster 1; points 3,4 belong to cluster 2.

5.4 Clustering of Patterns

5.4.1 Feature Space

Each pattern is a vector in feature space:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in \mathbb{R}^d$$

5.4.2 Similarity Measures

Common similarity measures:

Euclidean distance:

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{d} (x_{im} - x_{jm})^2}$$

Cosine similarity:

$$s(x_i, x_j) = \frac{x_i \cdot x_j}{\|x_i\| \|x_j\|}$$

Mahalanobis distance:

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T S^{-1} (x_i - x_j)}$$

where S is the covariance matrix.

5.5 Hierarchical Clustering

5.5.1 Distance Between Clusters

For hierarchical clustering, we need distance between clusters:

Table 5.1: Linkage Methods for Hierarchical Clustering

Method	**Formula**	**Properties**
Single Linkage	$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$	Can form chains, sensitive to noise
Complete Linkage	$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y)$	Produces compact clusters, sensitive to outliers
Average Linkage	$d(C_i, C_j) = \frac{1}{\lvert C_i \rvert \lvert C_j \rvert} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$	Compromise between single and complete
Centroid Linkage	$d(C_i, C_j) = \lVert \mu_i - \mu_j \rVert$	Uses cluster centers
Ward's Method	$d(C_i, C_j) = \frac{\lvert C_i \rvert \lvert C_j \rvert}{\lvert C_i \rvert + \lvert C_j \rvert} \lVert \mu_i - \mu_j \rVert^2$	Minimizes variance increase

Algorithm 4 Divisive Clustering (DIANA)

Require: Dataset X, desired number of clusters k
Ensure: Hierarchical partition of data
1: Start with all points in one cluster $C = X$
2: Initialize list of clusters $L = \{C\}$
3: **while** $|L| < k$ **do**
4: Select cluster $C_{\max}$ with largest diameter
5: Find the most dissimilar point in $C_{\max}$ (splinter group)
6: Reassign points to either splinter or original group based on proximity
7: Replace $C_{\max}$ with the two new clusters in L
8: **end while**
9: **return** L

Algorithm 5 Agglomerative Clustering (AGNES)

Require: Dataset $X = \{x_1, \ldots, x_n\}$, linkage function d_{link}
Ensure: Hierarchical partition of data
1: Initialize each point as its own cluster: $C_i = \{x_i\}$ for $i = 1, \ldots, n$
2: Initialize proximity matrix P with $P_{ij} = d(x_i, x_j)$
3: **while** number of clusters > 1 **do**
4: Find pair of clusters (C_i, C_j) with minimum $d_{link}(C_i, C_j)$
5: Merge C_i and C_j into $C_{\text{new}} = C_i \cup C_j$
6: Update proximity matrix using linkage function
7: **end while**
8: **return** dendrogram

5.5.2 Divisive Clustering (Top-Down)

Algorithm

5.5.3 Agglomerative Clustering (Bottom-Up)

Algorithm

5.5.4 Example Problem 5.2: Agglomerative Clustering

Problem: Given points in 1D: $\{1, 2, 5, 6, 9\}$, perform agglomerative clustering using single linkage. Show the dendrogram.

Solution:

Initial distances (Euclidean):

$$d(1,2) = 1, d(1,5) = 4, d(1,6) = 5, d(1,9) = 8$$

$$d(2,5) = 3, d(2,6) = 4, d(2,9) = 7$$

$$d(5,6) = 1, d(5,9) = 4, d(6,9) = 3$$

Step 1: Merge closest points (1,2) with distance 1, and (5,6) with distance 1. Clusters: $C_1 = \{1, 2\}$, $C_2 = \{5, 6\}$, $C_3 = \{9\}$

Step 2: Compute inter-cluster distances (single linkage):

$$d(C_1, C_2) = \min(d(1,5), d(1,6), d(2,5), d(2,6)) = \min(4, 5, 3, 4) = 3$$

$$d(C_1, C_3) = \min(d(1,9), d(2,9)) = \min(8, 7) = 7$$

$$d(C_2, C_3) = \min(d(5,9), d(6,9)) = \min(4, 3) = 3$$

Step 3: Merge closest clusters: either C_1 with C_2 or C_2 with C_3 (both distance 3). Choose C_2 and C_3: New cluster $C_4 = \{5, 6, 9\}$, remaining $C_1 = \{1, 2\}$

Step 4: Distance between C_1 and C_4:

$$d(C_1, C_4) = \min(d(1,5), d(1,6), d(1,9), d(2,5), d(2,6), d(2,9)) = \min(4, 5, 8, 3, 4, 7) = 3$$

Step 5: Merge C_1 and C_4 with distance 3.

Final dendrogram shows hierarchical structure.

5.6 Partitional Clustering

5.6.1 K-Means Clustering

Mathematical Formulation

K-means minimizes the within-cluster sum of squares:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} r_{ij} \|x_i - \mu_j\|^2$$

where:

$$r_{ij} = \begin{cases} 1 & \text{if } x_i \text{ belongs to cluster } j \\ 0 & \text{otherwise} \end{cases}$$

and $\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$ is the centroid of cluster j.

Algorithm

Algorithm 6 K-Means Clustering Algorithm

Require: Dataset $X = \{x_1, \ldots, x_n\}$, number of clusters k, maximum iterations T
Ensure: Cluster assignments and centroids

```
Initialize centroids μ_1, μ_2, ..., μ_k randomly
for t = 1 to T do
    Assignment Step:
    for i = 1 to n do
        c_i = arg min_j ||x_i − μ_j||^2
    end for
    Update Step:
    for j = 1 to k do
        μ_j = (1/|C_j|) Σ_{x_i ∈ C_j} x_i
    end for
    if centroids unchanged then
        break
    end if
end for
return {c_i}_{i=1}^n, {μ_j}_{j=1}^k
```

5.6.2 Convergence Properties

Theorem 5.6.1. *The K-means algorithm monotonically decreases the objective function J and converges to a local optimum in a finite number of iterations.*

5.6.3 Example Problem 5.3: K-Means Clustering

Problem: Given points in 2D: $A = (1, 1)$, $B = (2, 1)$, $C = (4, 3)$, $D = (5, 4)$. Perform K-means with $k = 2$, initial centroids $\mu_1 = (1, 1)$, $\mu_2 = (5, 4)$.

Solution:

Iteration 1:

Compute distances to centroids:

$$d(A, \mu_1) = \|(1, 1) - (1, 1)\| = 0, \quad d(A, \mu_2) = \|(1, 1) - (5, 4)\| = \sqrt{16 + 9} = 5$$

$$d(B, \mu_1) = \|(2, 1) - (1, 1)\| = 1, \quad d(B, \mu_2) = \|(2, 1) - (5, 4)\| = \sqrt{9 + 9} = \sqrt{18} \approx 4.24$$

$$d(C, \mu_1) = \|(4,3) - (1,1)\| = \sqrt{9+4} = \sqrt{13} \approx 3.61$$
$$d(C, \mu_2) = \|(4,3) - (5,4)\| = \sqrt{1+1} = \sqrt{2} \approx 1.41$$
$$d(D, \mu_1) = \|(5,4) - (1,1)\| = \sqrt{16+9} = 5$$
$$d(D, \mu_2) = \|(5,4) - (5,4)\| = 0$$

Assignment: Cluster 1 (μ_1): A, B Cluster 2 (μ_2): C, D
Update centroids:

$$\mu_1^{\text{new}} = \left(\frac{1+2}{2}, \frac{1+1}{2}\right) = (1.5, 1)$$
$$\mu_2^{\text{new}} = \left(\frac{4+5}{2}, \frac{3+4}{2}\right) = (4.5, 3.5)$$

Iteration 2:
Recalculate distances:

$$d(A, \mu_1^{\text{new}}) = \|(1,1) - (1.5,1)\| = 0.5$$
$$d(A, \mu_2^{\text{new}}) = \|(1,1) - (4.5,3.5)\| = \sqrt{12.25+6.25} = \sqrt{18.5} \approx 4.30$$
$$d(B, \mu_1^{\text{new}}) = \|(2,1) - (1.5,1)\| = 0.5$$
$$d(B, \mu_2^{\text{new}}) = \|(2,1) - (4.5,3.5)\| = \sqrt{6.25+6.25} = \sqrt{12.5} \approx 3.54$$
$$d(C, \mu_1^{\text{new}}) = \|(4,3) - (1.5,1)\| = \sqrt{6.25+4} = \sqrt{10.25} \approx 3.20$$
$$d(C, \mu_2^{\text{new}}) = \|(4,3) - (4.5,3.5)\| = \sqrt{0.25+0.25} = \sqrt{0.5} \approx 0.71$$
$$d(D, \mu_1^{\text{new}}) = \|(5,4) - (1.5,1)\| = \sqrt{12.25+9} = \sqrt{21.25} \approx 4.61$$
$$d(D, \mu_2^{\text{new}}) = \|(5,4) - (4.5,3.5)\| = \sqrt{0.25+0.25} = \sqrt{0.5} \approx 0.71$$

Assignment unchanged: Cluster 1: A,B; Cluster 2: C,D
Centroids unchanged: $\mu_1 = (1.5, 1)$, $\mu_2 = (4.5, 3.5)$
Algorithm converges.

5.7 Soft Partitioning and Soft Clustering

5.7.1 Fuzzy C-Means Clustering

Mathematical Formulation

Fuzzy C-means allows each point to belong to multiple clusters with membership degrees:

$$J_m = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^m \|x_i - c_j\|^2$$

where:

- $u_{ij} \in [0, 1]$ is the membership of point x_i in cluster j

- $m > 1$ is the fuzzification parameter
- c_j is the centroid of cluster j
- $\sum_{j=1}^{k} u_{ij} = 1$ for all i

Update Equations

Minimizing J_m subject to constraints yields:

Membership updates:

$$u_{ij} = \frac{1}{\sum_{l=1}^{k} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|} \right)^{\frac{2}{m-1}}}$$

Centroid updates:

$$c_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m}$$

5.7.2 Example Problem 5.4: Fuzzy C-Means

Problem: For point $x = (2, 3)$ and two clusters with centroids $c_1 = (1, 1)$, $c_2 = (4, 4)$, $m = 2$, compute membership values.

Solution:

Compute distances:

$$d_1 = \|(2, 3) - (1, 1)\| = \sqrt{1 + 4} = \sqrt{5} \approx 2.236$$

$$d_2 = \|(2, 3) - (4, 4)\| = \sqrt{4 + 1} = \sqrt{5} \approx 2.236$$

For $m = 2$, $\frac{2}{m-1} = 2$:

$$u_{11} = \frac{1}{\left(\frac{d_1}{d_1}\right)^2 + \left(\frac{d_1}{d_2}\right)^2} = \frac{1}{1 + (2.236/2.236)^2} = \frac{1}{1+1} = 0.5$$

$$u_{12} = \frac{1}{\left(\frac{d_2}{d_1}\right)^2 + \left(\frac{d_2}{d_2}\right)^2} = \frac{1}{(2.236/2.236)^2 + 1} = \frac{1}{1+1} = 0.5$$

The point belongs equally to both clusters.

5.7.3 Rough Clustering

Rough Sets Theory

In rough clustering, each cluster is represented by:

- **Lower approximation** $\underline{C_j}$: Points definitely belonging to cluster j
- **Upper approximation** $\overline{C_j}$: Points possibly belonging to cluster j

- **Boundary region** $BN(C_j) = \overline{C_j} - \underline{C_j}$: Points with uncertain membership

Properties:

$$\underline{C_j} \subseteq C_j \subseteq \overline{C_j}$$

$$\underline{C_j} \cap \underline{C_l} = \emptyset \quad \text{for } j \neq l$$

5.7.4 Rough K-Means Clustering Algorithm

Algorithm Description

Algorithm 7 Rough K-Means Clustering Algorithm

Require: Dataset X, number of clusters k, thresholds T_{low} and T_{high}, relative importance w_{low}, w_{high}

Ensure: Rough clusters with lower and upper approximations

1: Initialize centroids $\mu_1, \mu_2, \ldots, \mu_k$ randomly
2: **repeat**
3: For each point x_i, compute distances to all centroids $d_{ij} = \|x_i - \mu_j\|$
4: Let $d_{\min} = \min_j d_{ij}$
5: Let $T = \{j : d_{ij} - d_{\min} \leq T_{low}\}$
6: **if** $|T| = 1$ **then**
7: Assign x_i to lower approximation of cluster $j \in T$
8: **else**
9: Assign x_i to upper approximations of all clusters in T
10: **end if**
11: Update centroids:

$$\mu_j = w_{low} \cdot \frac{\sum_{x_i \in \underline{C_j}} x_i}{|\underline{C_j}|} + w_{high} \cdot \frac{\sum_{x_i \in \overline{C_j}} x_i}{|\overline{C_j}|}$$

12: **until** convergence
13: **return** Lower and upper approximations for each cluster

5.8 Expectation Maximization-Based Clustering

5.8.1 Gaussian Mixture Models (GMM)

Data is modeled as a mixture of k Gaussian distributions:

$$p(x) = \sum_{j=1}^{k} \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)$$

where π_j are mixing coefficients with $\sum_{j=1}^{k} \pi_j = 1$, $\pi_j \geq 0$.

5.8.2 EM Algorithm for GMM

Expectation Step (E-Step)

Compute responsibilities (posterior probabilities):

$$\gamma_{ij} = \frac{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l, \Sigma_l)}$$

Maximization Step (M-Step)

Update parameters:

Mixing coefficients:

$$\pi_j^{\text{new}} = \frac{1}{n} \sum_{i=1}^{n} \gamma_{ij}$$

Means:

$$\mu_j^{\text{new}} = \frac{\sum_{i=1}^{n} \gamma_{ij} x_i}{\sum_{i=1}^{n} \gamma_{ij}}$$

Covariances:

$$\Sigma_j^{\text{new}} = \frac{\sum_{i=1}^{n} \gamma_{ij} (x_i - \mu_j^{\text{new}})(x_i - \mu_j^{\text{new}})^T}{\sum_{i=1}^{n} \gamma_{ij}}$$

5.8.3 Example Problem 5.5: EM Algorithm

Problem: Given points $\{1, 2, 8, 9\}$ with initial parameters: Cluster 1: $\pi_1 = 0.5$, $\mu_1 = 1.5$, $\sigma_1^2 = 1$ Cluster 2: $\pi_2 = 0.5$, $\mu_2 = 8.5$, $\sigma_2^2 = 1$ Perform one E-step and M-step.

Solution:

E-Step:

For $x = 1$:

$$\mathcal{N}(1|1.5, 1) = \frac{1}{\sqrt{2\pi}} e^{-(1-1.5)^2/2} = 0.352$$

$$\mathcal{N}(1|8.5, 1) = \frac{1}{\sqrt{2\pi}} e^{-(1-8.5)^2/2} \approx 0$$

$$\gamma_{11} = \frac{0.5 \times 0.352}{0.5 \times 0.352 + 0.5 \times 0} = 1$$

$$\gamma_{21} = 0$$

For $x = 2$:

$$\mathcal{N}(2|1.5, 1) = \frac{1}{\sqrt{2\pi}} e^{-(2-1.5)^2/2} = 0.352$$

$$\mathcal{N}(2|8.5, 1) = \frac{1}{\sqrt{2\pi}} e^{-(2-8.5)^2/2} \approx 0$$

$$\gamma_{12} = 1, \gamma_{22} = 0$$

For $x = 8$:

$$\mathcal{N}(8|1.5, 1) \approx 0, \quad \mathcal{N}(8|8.5, 1) = 0.352$$

$$\gamma_{13} = 0, \gamma_{23} = 1$$

For $x = 9$:

$$\mathcal{N}(9|1.5, 1) \approx 0, \quad \mathcal{N}(9|8.5, 1) = 0.352$$

$$\gamma_{14} = 0, \gamma_{24} = 1$$

M-Step:

Cluster 1:

$$n_1 = \sum \gamma_{1i} = 1 + 1 + 0 + 0 = 2$$

$$\pi_1^{\text{new}} = 2/4 = 0.5$$

$$\mu_1^{\text{new}} = (1 \times 1 + 1 \times 2)/2 = 1.5$$

$$\sigma_1^{2,\text{new}} = [1 \times (1 - 1.5)^2 + 1 \times (2 - 1.5)^2]/2 = (0.25 + 0.25)/2 = 0.25$$

Cluster 2:

$$n_2 = \sum \gamma_{2i} = 0 + 0 + 1 + 1 = 2$$

$$\pi_2^{\text{new}} = 2/4 = 0.5$$

$$\mu_2^{\text{new}} = (1 \times 8 + 1 \times 9)/2 = 8.5$$

$$\sigma_2^{2,\text{new}} = [1 \times (8 - 8.5)^2 + 1 \times (9 - 8.5)^2]/2 = (0.25 + 0.25)/2 = 0.25$$

Parameters remain the same (converged).

5.9 Spectral Clustering

5.9.1 Graph-Based Formulation

Construct a similarity graph $G = (V, E)$ where vertices are data points and edge weights represent similarity.

5.9.2 Similarity Matrix

Gaussian similarity function:

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

5.9.3 Graph Laplacian

Unnormalized Laplacian:

$$L = D - W$$

where D is degree matrix with $D_{ii} = \sum_j W_{ij}$.

Normalized Laplacian (symmetric):

$$L_{\text{sym}} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$$

Normalized Laplacian (random walk):

$$L_{\text{rw}} = D^{-1} L = I - D^{-1} W$$

5.9.4 Spectral Clustering Algorithm

Algorithm 8 Spectral Clustering

Require: Dataset $X = \{x_1, \ldots, x_n\}$, number of clusters k, similarity parameter σ
Ensure: Cluster assignments

1: Compute similarity matrix W with $W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
2: Compute degree matrix D with $D_{ii} = \sum_j W_{ij}$
3: Compute Laplacian $L = D - W$
4: Compute first k eigenvectors $u_1, \ldots, u_k$ of L corresponding to smallest eigenvalues
5: Form matrix $U \in \mathbb{R}^{n \times k}$ with $u_1, \ldots, u_k$ as columns
6: For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the i-th row of U
7: Cluster points $\{y_i\}_{i=1}^n$ using K-means into k clusters
8: **return** Cluster assignments

5.9.5 Example Problem 5.6: Spectral Clustering

Problem: For points in 2D: $A = (0, 0)$, $B = (0.1, 0)$, $C = (1, 1)$, $D = (1.1, 1)$. Use $\sigma = 0.5$ to compute similarity matrix and perform spectral clustering with $k = 2$.

Solution:

Compute distances:

$$d(A, B) = 0.1, d(A, C) = \sqrt{2} \approx 1.414, d(A, D) = \sqrt{1.21 + 1} \approx 1.49$$

$$d(B, C) = \sqrt{0.81 + 1} \approx 1.345, d(B, D) = \sqrt{1 + 1} \approx 1.414$$

$$d(C, D) = 0.1$$

Similarity matrix ($W_{ij} = e^{-d_{ij}^2/(2\times 0.25)} = e^{-2d_{ij}^2}$):

$$W = \begin{bmatrix} 1 & e^{-0.02} \approx 0.98 & e^{-4} \approx 0.018 & e^{-4.44} \approx 0.012 \\ 0.98 & 1 & e^{-3.62} \approx 0.027 & e^{-4} \approx 0.018 \\ 0.018 & 0.027 & 1 & e^{-0.02} \approx 0.98 \\ 0.012 & 0.018 & 0.98 & 1 \end{bmatrix}$$

Degree matrix D (diagonal sums):

$$D_{11} = 1 + 0.98 + 0.018 + 0.012 = 2.01$$

$$D_{22} = 0.98 + 1 + 0.027 + 0.018 = 2.025$$

$$D_{33} = 0.018 + 0.027 + 1 + 0.98 = 2.025$$

$$D_{44} = 0.012 + 0.018 + 0.98 + 1 = 2.01$$

Laplacian $L = D - W$:

$$L \approx \begin{bmatrix} 1.01 & -0.98 & -0.018 & -0.012 \\ -0.98 & 1.025 & -0.027 & -0.018 \\ -0.018 & -0.027 & 1.025 & -0.98 \\ -0.012 & -0.018 & -0.98 & 1.01 \end{bmatrix}$$

Eigenvalues and eigenvectors (approximate):

$$\lambda_1 \approx 0, v_1 \approx [0.5, 0.5, 0.5, 0.5]^T$$

$$\lambda_2 \approx 0.04, v_2 \approx [0.5, 0.5, -0.5, -0.5]^T$$

$$\lambda_3 \approx 2, \lambda_4 \approx 2$$

Using v_2 for clustering: Points with positive entries (A,B) form cluster 1 Points with negative entries (C,D) form cluster 2

5.10 Comparison of Clustering Methods

Table 5.2: Comparison of Clustering Methods

Method	**Advantages**	**Disadvantages**	**Best For**
K-Means	Simple, fast, scalable	Assumes spherical clusters, need k	Large datasets, convex clusters
Hierarchical	No need to specify k, dendrogram	$O(n^3)$ complexity, irreversible	Small datasets, hierarchical structure
Fuzzy C-Means	Soft assignments, uncertainty	Sensitive to m, computationally heavy	Overlapping clusters
EM/GMM	Probabilistic, flexible shapes	Can overfit, slow convergence	Elliptical clusters, density estimation
Spectral	Non-convex clusters, arbitrary shapes	Need similarity matrix, k selection	Complex cluster shapes, graph data
Rough Sets	Handles uncertainty, interpretable	Threshold selection, limited theory	Data with boundary regions

5.11 Evaluation of Clustering

5.11.1 Internal Validation Measures

Silhouette coefficient:

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

where $a(i)$ is mean distance to points in same cluster, $b(i)$ is mean distance to points in nearest cluster.

Davies-Bouldin index:

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)} \right)$$

where σ_i is average distance within cluster i.

5.11.2 External Validation Measures

Rand index:

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

Adjusted Rand Index:

$$ARI = \frac{RI - \mathbb{E}[RI]}{\max(RI) - \mathbb{E}[RI]}$$

5.12 Multiple Choice Questions

5.12.1 Knowledge Level Questions

1. Which clustering method requires specifying the number of clusters in advance?
   (a) Hierarchical clustering
   (b) **K-Means clustering** (Correct)
   (c) DBSCAN
   (d) All of the above

2. In fuzzy C-means, the fuzzification parameter m controls:
   (a) Number of clusters
   (b) **Degree of fuzziness in membership** (Correct)
   (c) Convergence rate
   (d) Cluster shape

5.12.2 Comprehension Level Questions

3. The EM algorithm for GMM clustering alternates between:
   (a) Assignment and update steps
   (b) **Expectation and maximization steps** (Correct)
   (c) Forward and backward passes
   (d) Training and validation

4. In spectral clustering, the eigenvectors of which matrix are used?
   (a) Data matrix

(b) Covariance matrix

(c) **Graph Laplacian** (Correct)

(d) Similarity matrix

5.12.3 Application Level Questions

5. For K-means with $k = 3$, if one cluster becomes empty during iteration, what should happen?

   (a) Algorithm terminates

   (b) Empty cluster is removed

   (c) **Empty cluster is reinitialized** (Correct)

   (d) Error is raised

6. Given points forming two concentric circles, which clustering method would work best?

   (a) K-Means

   (b) Hierarchical

   (c) **Spectral clustering** (Correct)

   (d) Fuzzy C-means

5.12.4 Analysis Level Questions

7. The main limitation of K-means is:

   (a) High computational complexity

   (b) **Assumes spherical clusters of equal size** (Correct)

   (c) Cannot handle numeric data

   (d) Requires distance matrix

8. In agglomerative clustering, single linkage tends to:

   (a) Produce compact clusters

   (b) **Form chain-like clusters** (Correct)

   (c) Be robust to outliers

   (d) Require $O(n^2)$ memory

5.13 Essay Questions and Solutions

1. **Question**: Derive the update equations for K-means clustering and prove its convergence.

   Solution:

   Objective function:

   $$J(\{r_{ij}\}, \{\mu_j\}) = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij} \|x_i - \mu_j\|^2$$

   where $r_{ij} \in \{0, 1\}$ and $\sum_j r_{ij} = 1$.

   Optimization:

   For fixed μ_j, optimal assignment:

   $$r_{ij} = \begin{cases} 1 & \text{if } j = \arg\min_l \|x_i - \mu_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

   For fixed r_{ij}, optimal μ_j:

   $$\frac{\partial J}{\partial \mu_j} = -2 \sum_{i=1}^{n} r_{ij}(x_i - \mu_j) = 0$$

   $$\sum_{i=1}^{n} r_{ij} x_i = \mu_j \sum_{i=1}^{n} r_{ij}$$

   $$\mu_j = \frac{\sum_{i=1}^{n} r_{ij} x_i}{\sum_{i=1}^{n} r_{ij}}$$

   Convergence proof:

   (a) Assignment step minimizes J for fixed centroids

   (b) Update step minimizes J for fixed assignments

   (c) J is bounded below (non-negative)

   (d) Each step non-increases J

   (e) Finite number of possible assignments (k^n)

   (f) Algorithm converges to local minimum

2. **Question**: Explain the EM algorithm for Gaussian Mixture Models with detailed mathematical derivations.

   Solution:

Complete data log-likelihood:

$$\log p(X, Z|\theta) = \sum_{i=1}^{n} \sum_{j=1}^{k} z_{ij} [\log \pi_j + \log \mathcal{N}(x_i|\mu_j, \Sigma_j)]$$

E-Step: Compute expected value of z_{ij}:

$$\gamma_{ij} = \mathbb{E}[z_{ij}|X, \theta] = \frac{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}{\sum_{l=1}^{k} \pi_l \mathcal{N}(x_i|\mu_l, \Sigma_l)}$$

Q-function:

$$Q(\theta|\theta^{(t)}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} [\log \pi_j + \log \mathcal{N}(x_i|\mu_j, \Sigma_j)]$$

M-Step: Maximize Q with respect to parameters:

For π_j with constraint $\sum \pi_j = 1$:

$$\pi_j = \frac{1}{n} \sum_{i=1}^{n} \gamma_{ij}$$

For μ_j:

$$\frac{\partial Q}{\partial \mu_j} = \sum_{i=1}^{n} \gamma_{ij} \Sigma_j^{-1} (x_i - \mu_j) = 0$$

$$\mu_j = \frac{\sum_{i=1}^{n} \gamma_{ij} x_i}{\sum_{i=1}^{n} \gamma_{ij}}$$

For Σ_j:

$$\Sigma_j = \frac{\sum_{i=1}^{n} \gamma_{ij} (x_i - \mu_j)(x_i - \mu_j)^T}{\sum_{i=1}^{n} \gamma_{ij}}$$

3. **Question**: Compare and contrast hard clustering (K-means) with soft clustering (Fuzzy C-means and EM-GMM), discussing their mathematical formulations, advantages, and limitations.

   Solution:

   K-Means (Hard Clustering):

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij} \|x_i - \mu_j\|^2, \quad r_{ij} \in \{0, 1\}$$

   Advantages: Simple, fast, scalable Limitations: Hard assignments, spherical clusters

Fuzzy C-Means (Soft Clustering):

$$J_m = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^m \|x_i - c_j\|^2, \quad u_{ij} \in [0, 1]$$

Advantages: Membership degrees, handles uncertainty Limitations: Parameter m sensitive, still assumes spherical clusters

EM-GMM (Probabilistic Soft Clustering):

$$p(x) = \sum_{j=1}^{k} \pi_j \mathcal{N}(x|\mu_j, \Sigma_j)$$

Advantages: Probabilistic foundation, elliptical clusters Limitations: Can overfit, sensitive to initialization

Comparison:

- All minimize within-cluster scatter
- K-means is a special case of EM-GMM with spherical, equal covariance
- Fuzzy C-means relates to K-means with fuzzification
- EM provides probability estimates, others give membership degrees
- Choice depends on data characteristics and application requirements

5.14 Summary and Key Takeaways

5.14.1 Mathematical Foundations

- Clustering: $\min \sum_{j=1}^{k} \sum_{x_i \in C_j} d(x_i, \mu_j)^2$
- K-means alternates between assignment and update
- EM for GMM: $\gamma_{ij} = \frac{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}{\sum_l \pi_l \mathcal{N}(x_i|\mu_l, \Sigma_l)}$
- Spectral clustering uses eigenvectors of graph Laplacian
- Fuzzy C-means: $u_{ij} = 1/\sum_l (d_{ij}/d_{il})^{2/(m-1)}$

5.14.2 Important Formulas

K-means centroid: $\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$

GMM responsibility: $\gamma_{ij} = \frac{\pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}{\sum_l \pi_l \mathcal{N}(x_i|\mu_l, \Sigma_l)}$

Fuzzy membership: $u_{ij} = \left[\sum_{l=1}^{k} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|} \right)^{\frac{2}{m-1}} \right]^{-1}$

Graph Laplacian: $L = D - W$

Silhouette: $s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$

5.14.3 Practical Guidelines

1. Scale features before clustering
2. Use multiple initializations for K-means
3. Elbow method for choosing k
4. Silhouette analysis for cluster validation
5. Consider data shape when choosing algorithm
6. For non-convex clusters, use spectral clustering
7. For overlapping clusters, use soft clustering

Note: Clustering is fundamental to unsupervised learning. Understanding the mathematical foundations enables proper algorithm selection and parameter tuning for specific applications.