

COMPREHENSIVE EXAM NOTES

Algorithms & Machine Learning

Unit	Topic	Page
I	Introduction to Algorithms and Machine Learning	2
II	Algorithms: Graphs, DP, Hashing	8
III	Application to Personal Genomics	15
IV	Machine Learning Fundamentals	20
V	Machine Learning Applications	27

UNIT I: Introduction to Algorithms and Machine Learning

1.1 What is an Algorithm?

Definition: An algorithm is a well-defined, finite sequence of instructions to solve a computational problem. It takes input, processes it through a series of steps, and produces output.

Key Characteristics of Algorithms:

- **Input:** Zero or more inputs from a specified set
- **Output:** One or more outputs with a specified relation to inputs
- **Definiteness:** Each step is precisely and unambiguously defined
- **Finiteness:** Algorithm terminates after finite number of steps
- **Effectiveness:** Each step is basic enough to be carried out

1.2 Tools to Analyze Algorithms

1.2.1 Time Complexity Analysis

Time complexity measures the amount of time an algorithm takes as a function of input size. We use asymptotic notation to describe growth rates.

Big-O Notation (Upper Bound):

$$f(n) = O(g(n)) \text{ if } \exists \text{ constants } c > 0, n_0 > 0 \text{ such that:}$$
$$0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0$$

Big-Ω Notation (Lower Bound):

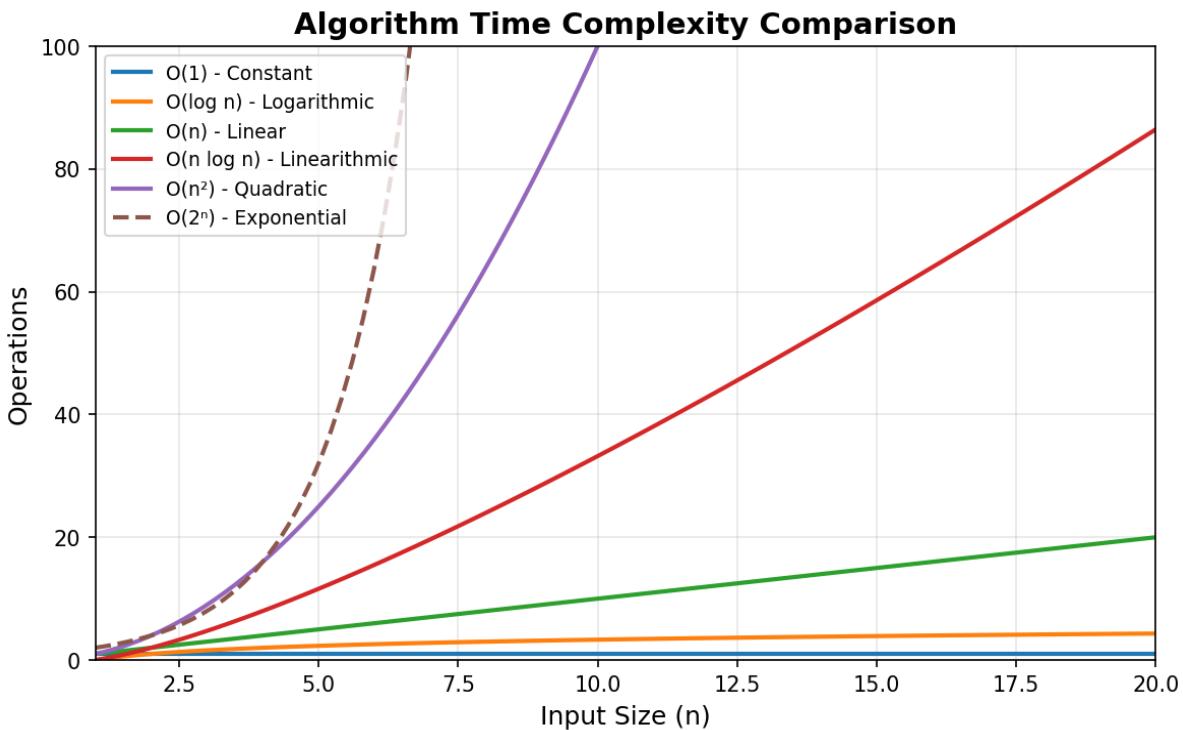
$$f(n) = \Omega(g(n)) \text{ if } \exists \text{ constants } c > 0, n_0 > 0 \text{ such that:}$$
$$0 \leq c \cdot g(n) \leq f(n) \text{ for all } n \geq n_0$$

Big-Θ Notation (Tight Bound):

$$f(n) = \Theta(g(n)) \text{ if } f(n) = O(g(n)) \text{ AND } f(n) = \Omega(g(n))$$

Complexity	Name	Example	n=1000
O(1)	Constant	Array access	1
O(log n)	Logarithmic	Binary search	10
O(n)	Linear	Linear search	1000
O(n log n)	Linearithmic	Merge sort	10000
O(n ²)	Quadratic	Bubble sort	1000000

$O(2^n)$	Exponential	Subset generation	$\approx 10^3$
----------	-------------	-------------------	----------------



1.2.2 Space Complexity

Space complexity measures memory usage as a function of input size. Total Space = Input Space + Auxiliary Space

1.2.3 Recurrence Relations

Recurrence relations express algorithm complexity recursively. Common methods to solve them:

- **Substitution Method:** Guess solution, prove by induction
- **Recursion Tree Method:** Visualize recursive calls as tree
- **Master Theorem:** For recurrences of form $T(n) = aT(n/b) + f(n)$

Master Theorem:

$$T(n) = aT(n/b) + f(n) \text{ where } a \geq 1, b > 1$$

Case 1: If $f(n) = O(n^{\log_b(a) - \epsilon})$ for $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b(a)})$

Case 2: If $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \cdot \log n)$

Case 3: If $f(n) = \Omega(n^{\log_b(a) + \epsilon})$ for $\epsilon > 0$, then $T(n) = \Theta(f(n))$

1.3 Divide and Conquer

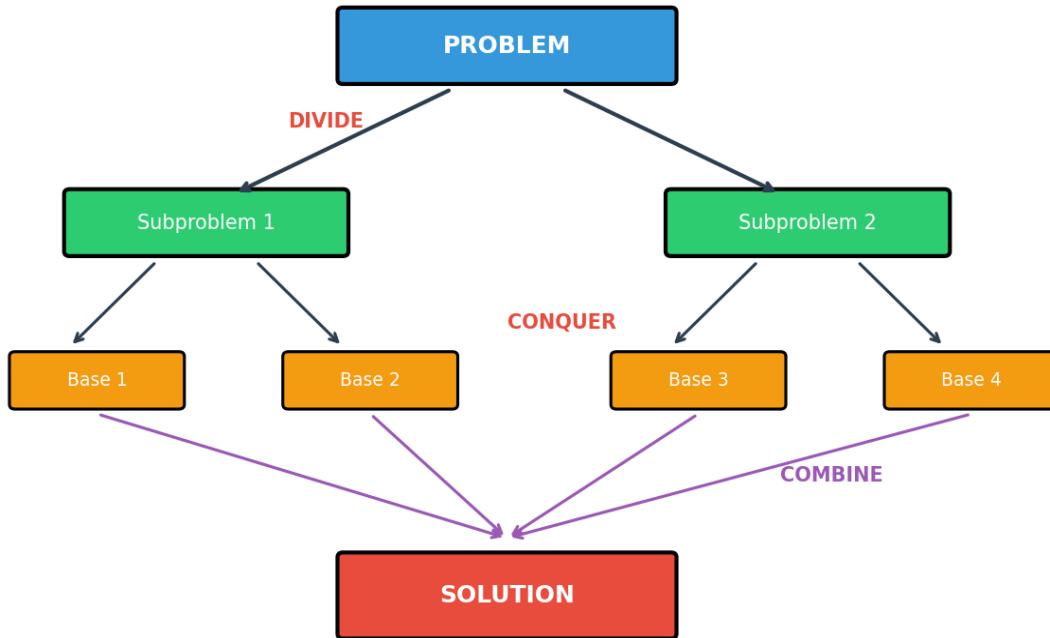
Divide and Conquer is a paradigm where a problem is broken into smaller subproblems, solved recursively, and combined to get the final solution.

Three Steps:

1. **DIVIDE:** Break problem into smaller subproblems
2. **CONQUER:** Solve subproblems recursively (base case: solve directly)

3. COMBINE: Merge subproblem solutions into final solution

Divide and Conquer Paradigm



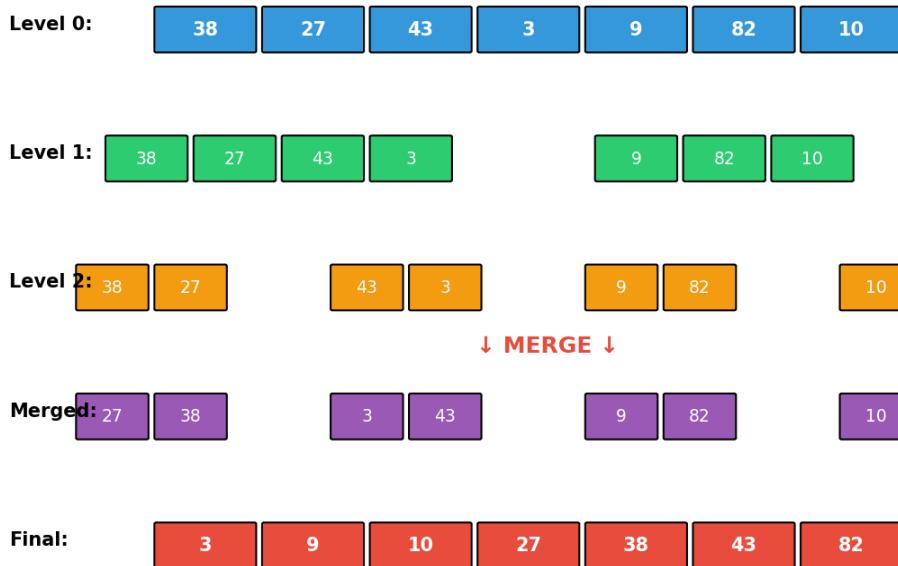
1.3.1 Merge Sort Example

Merge Sort divides array in half, recursively sorts each half, then merges sorted halves.

Time Complexity: $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$

Space Complexity: $O(n)$ auxiliary space

Merge Sort Algorithm Visualization



1.3.2 Quick Sort

Quick Sort selects a pivot, partitions array around pivot (smaller elements left, larger right), then recursively sorts partitions.

Average Case: $\Theta(n \log n)$ | Worst Case: $O(n^2)$ | Space: $O(\log n)$

1.4 Randomization in Algorithms

Randomized algorithms use random numbers to make decisions during execution, often providing better expected performance or simpler implementation.

Types of Randomized Algorithms:

- **Las Vegas:** Always correct, runtime varies (e.g., Randomized QuickSort)
- **Monte Carlo:** Fixed runtime, may have small error probability (e.g., Miller-Rabin primality)

Example: Randomized QuickSort

By randomly selecting the pivot element, we avoid worst-case $O(n^2)$ behavior on already sorted arrays. Expected running time becomes $O(n \log n)$ for all inputs.

1.5 Introduction to Machine Learning

Machine Learning is a subset of AI where systems learn patterns from data without being explicitly programmed. It focuses on building algorithms that improve through experience.

Types of Machine Learning:

Type	Description	Example
Supervised	Learn from labeled data	Classification, Regression

Unsupervised	Find patterns in unlabeled data	Clustering, Dimensionality Reduction
Reinforcement	Learn through rewards/penalties	Game playing, Robotics
Semi-supervised	Mix of labeled and unlabeled	Web content classification

UNIT II: Algorithms - Graphs, DP, Hashing, Search Trees

2.1 Graph Algorithms

A graph $G = (V, E)$ consists of vertices V and edges E . Graphs can be directed/undirected, weighted/unweighted, and are used to model networks, relationships, and paths.

Graph Representations:

- **Adjacency Matrix:** 2D array, $A[i][j] = 1$ if edge (i,j) exists. Space: $O(V^2)$
- **Adjacency List:** Array of lists, each vertex stores neighbors. Space: $O(V+E)$

2.1.1 Breadth-First Search (BFS)

BFS explores graph level by level using a queue. It finds shortest path in unweighted graphs.

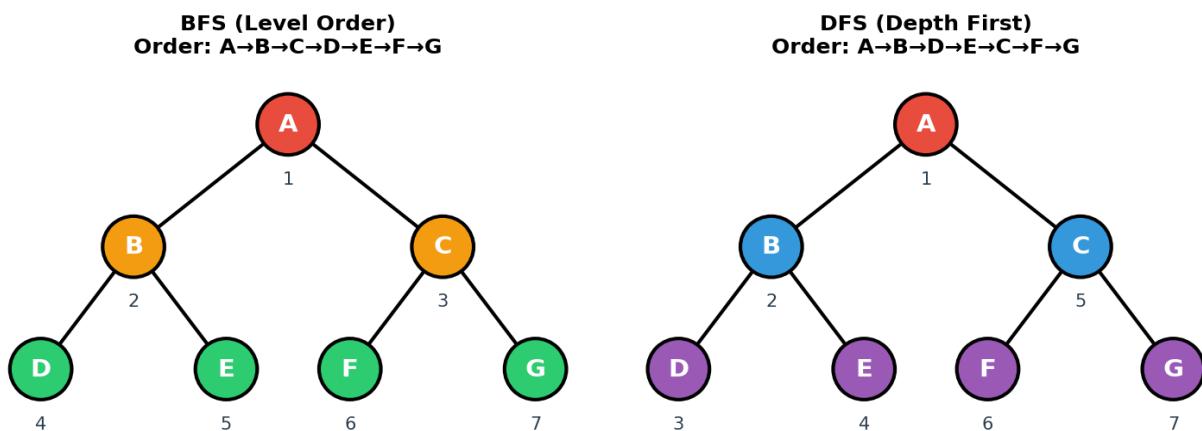
Time: $O(V + E)$ | Space: $O(V)$

2.1.2 Depth-First Search (DFS)

DFS explores as deep as possible before backtracking using stack/recursion. Used for topological sort, cycle detection, connected components.

Time: $O(V + E)$ | Space: $O(V)$

Graph Traversal Algorithms



2.1.3 Dijkstra's Shortest Path Algorithm

Finds shortest paths from source to all vertices in weighted graph (non-negative weights). Uses greedy approach with priority queue.

Time: $O((V + E) \log V)$ with binary heap | Space: $O(V)$

Algorithm Steps:

1. Initialize $\text{dist}[\text{source}] = 0$, $\text{dist}[\text{all others}] = \infty$

2. Add all vertices to priority queue
3. While queue not empty: extract min, relax all neighbors
4. Relaxation: if $\text{dist}[u] + \text{weight}(u,v) < \text{dist}[v]$, update $\text{dist}[v]$

2.2 Stable Marriage Problem

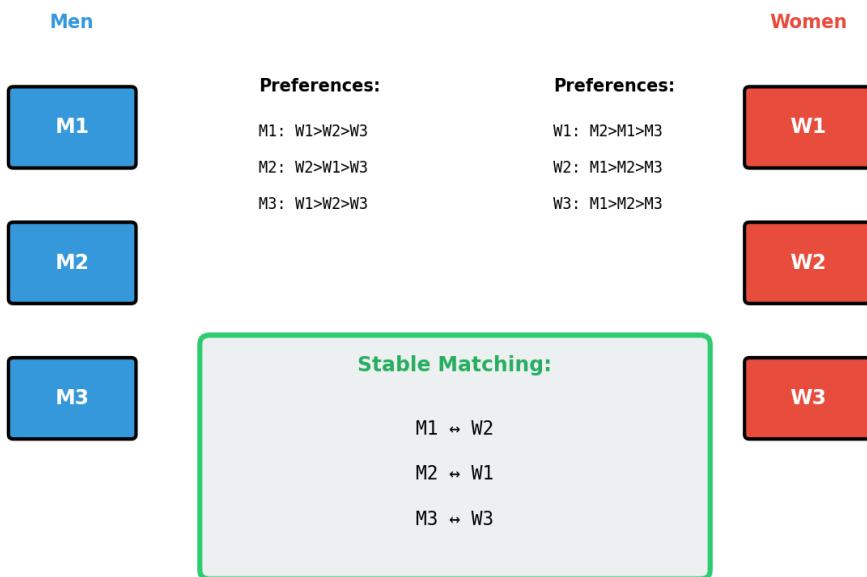
Given n men and n women with ranked preferences, find a stable matching where no two people prefer each other over their current partners.

Gale-Shapley Algorithm:

1. Each unengaged man proposes to highest-ranked woman not yet proposed to
2. Woman accepts if unengaged or prefers proposer over current partner
3. Repeat until all matched

Time Complexity: $O(n^2)$ | Result: Men-optimal stable matching

Gale-Shapley Algorithm: Stable Marriage Problem



No pair (m,w) exists where both prefer each other over current partners

2.3 Dictionaries and Hashing

A dictionary (hash table) stores key-value pairs with $O(1)$ average-case operations using hash functions to map keys to array indices.

Hash Function Properties:

- **Deterministic:** Same key always produces same hash
- **Uniform Distribution:** Keys spread evenly across buckets
- **Fast Computation:** $O(1)$ to compute hash

Common Hash Functions:

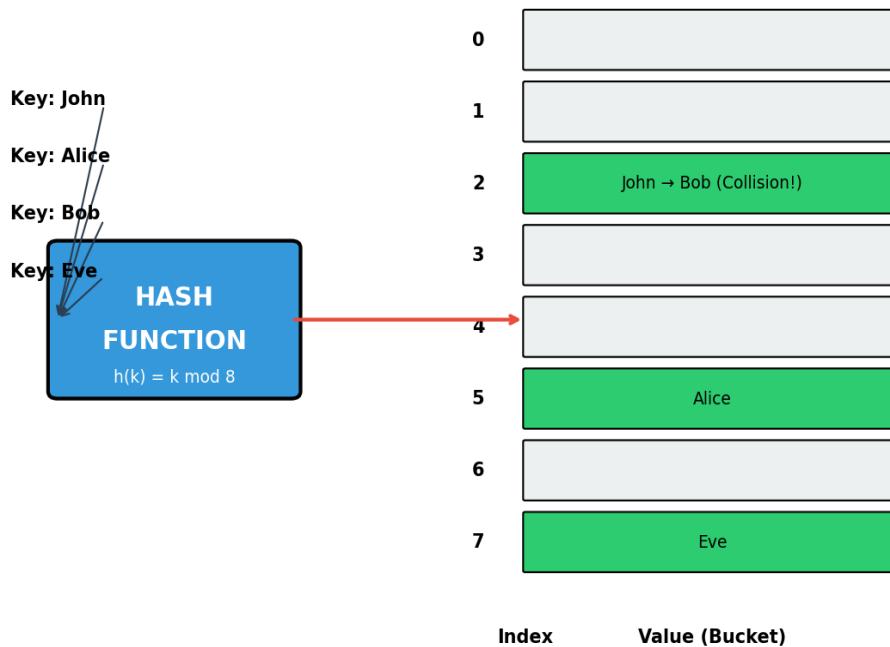
Division Method: $h(k) = k \bmod m$

Multiplication Method: $h(k) = \lceil m(kA \bmod 1) \rceil$ where $0 < A < 1$

Collision Resolution:

- **Chaining:** Store collisions in linked list at each bucket
- **Open Addressing:** Probe for next empty slot (linear, quadratic, double hashing)

Hash Table with Chaining for Collision Resolution



Operation	Average Case	Worst Case
Search	$O(1)$	$O(n)$
Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$

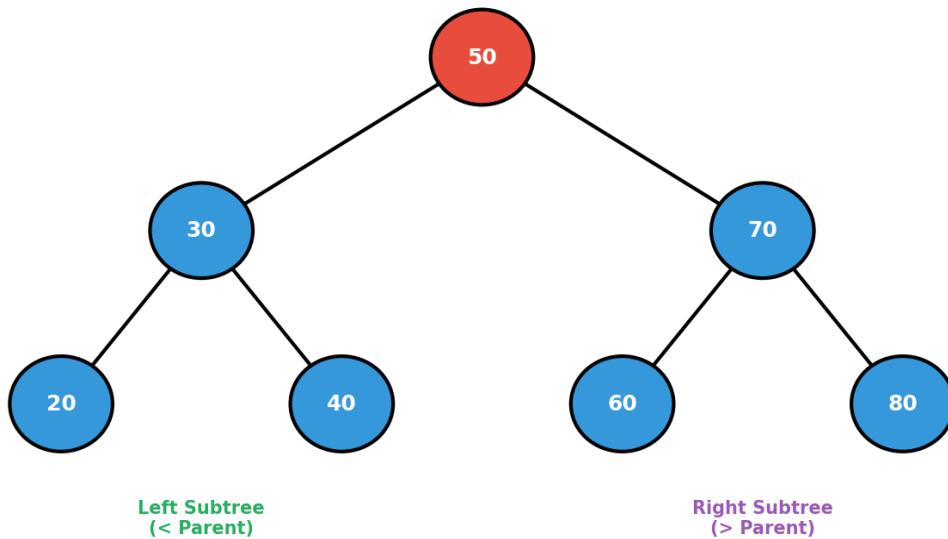
2.4 Search Trees

2.4.1 Binary Search Tree (BST)

A BST is a binary tree where left subtree contains smaller keys and right subtree contains larger keys.

BST Property: For each node x : $\text{left}(x) < x < \text{right}(x)$

Binary Search Tree (BST)



Operation	Average	Worst (Skewed)
Search	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$
Min/Max	$O(\log n)$	$O(n)$

2.4.2 Balanced Trees (AVL, Red-Black)

Self-balancing BSTs maintain $O(\log n)$ height through rotations after insertions/deletions.

- **AVL Tree:** Height difference of subtrees ≤ 1 for all nodes
- **Red-Black Tree:** Nodes colored red/black with balancing properties

Both guarantee $O(\log n)$ operations

2.5 Dynamic Programming

Dynamic Programming solves problems by breaking into overlapping subproblems, solving each once, and storing results to avoid redundant computation.

Key Properties for DP:

1. **Optimal Substructure:** Optimal solution contains optimal solutions to subproblems
2. **Overlapping Subproblems:** Same subproblems solved multiple times

Approaches:

- **Top-Down (Memoization):** Recursive with caching
- **Bottom-Up (Tabulation):** Iterative, fill table from base cases

Dynamic Programming: Fibonacci Sequence

$$F(n) = F(n-1) + F(n-2), \text{ Base: } F(0)=0, F(1)=1$$

Index:	0	1	2	3	4	5	6	7
Value:	0	1	1	2	3	5	8	13

$$\text{Example: } F(4) = F(3) + F(2) = 2 + 1 = 3$$

Time: O(n) | Space: O(n) or O(1) with optimization

Classic DP Problems:

Problem	Recurrence	Time	Space
Fibonacci	$F(n) = F(n-1) + F(n-2)$	$O(n)$	$O(1)$
Longest Common Subsequence	$LCS[i][j] = LCS[i-1][j-1]+1 \text{ or } \max(\dots)$	$O(mn)$	$O(mn)$
0/1 Knapsack	$K[i][w] = \max(K[i-1][w], v[i]+K[i-1][w-w[i]])$	$O(nW)$	$O(nW)$
Edit Distance	$E[i][j] = \min(E[i-1][j]+1, E[i][j-1]+1, \dots)$	$O(mn)$	$O(mn)$

UNIT III: Application to Personal Genomics

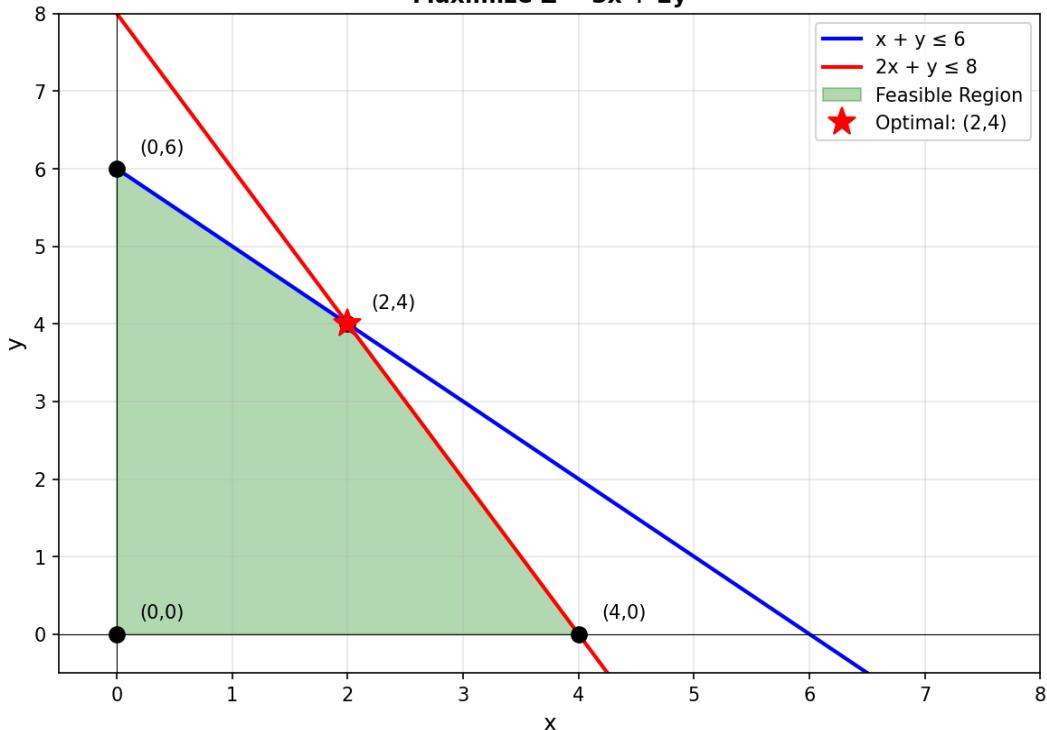
3.1 Linear Programming

Linear Programming (LP) optimizes a linear objective function subject to linear equality and inequality constraints. It's used in resource allocation, scheduling, and optimization problems.

Standard Form:

$$\begin{aligned} \text{Maximize: } Z &= c_1x_1 + c_2x_2 + \dots + c_nx_n \\ \text{Subject to: } a_{11}x_1 + a_{12}x_2 + \dots &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots &\leq b_2 \\ x_i &\geq 0 \text{ for all } i \end{aligned}$$

Linear Programming: Graphical Method
Maximize $Z = 3x + 2y$



Solution Methods:

- **Graphical Method:** For 2 variables, plot constraints, find feasible region, optimize at vertices
- **Simplex Method:** Iteratively move along edges of feasible polytope to optimal vertex
- **Interior Point Methods:** Move through interior of feasible region

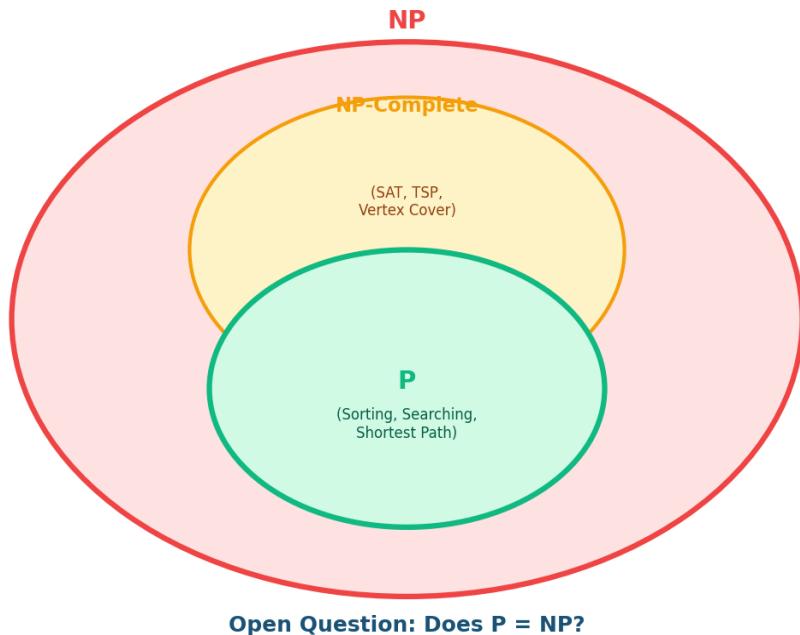
3.2 NP-Completeness

NP-Complete problems are the hardest problems in NP - if any one can be solved in polynomial time, then all NP problems can be ($P = NP$).

Complexity Classes:

- **P:** Problems solvable in polynomial time
- **NP:** Problems verifiable in polynomial time
- **NP-Hard:** At least as hard as hardest NP problems
- **NP-Complete:** In NP AND NP-Hard

Complexity Classes: P, NP, NP-Complete



Proving NP-Completeness:

1. Show problem is in NP (solution verifiable in polynomial time)
2. Reduce a known NP-Complete problem to it in polynomial time

Famous NP-Complete Problems:

Problem	Description
SAT	Boolean satisfiability - find assignment making formula true
3-SAT	SAT with clauses of exactly 3 literals
Traveling Salesman	Find shortest tour visiting all cities once
Vertex Cover	Find minimum set of vertices covering all edges
Subset Sum	Find subset summing to target value
Graph Coloring	Color graph with k colors, no adjacent same color

3.3 Introduction to Personal Genomics

Personal genomics analyzes individual genetic information to understand health risks, ancestry, drug responses, and inherited traits using computational methods.

Key Concepts:

- **Genome:** Complete DNA sequence (~3 billion base pairs in humans)
- **SNP (Single Nucleotide Polymorphism):** Single base variation between individuals

- **Variant:** Genetic difference from reference genome
- **Phenotype:** Observable characteristics determined by genotype + environment

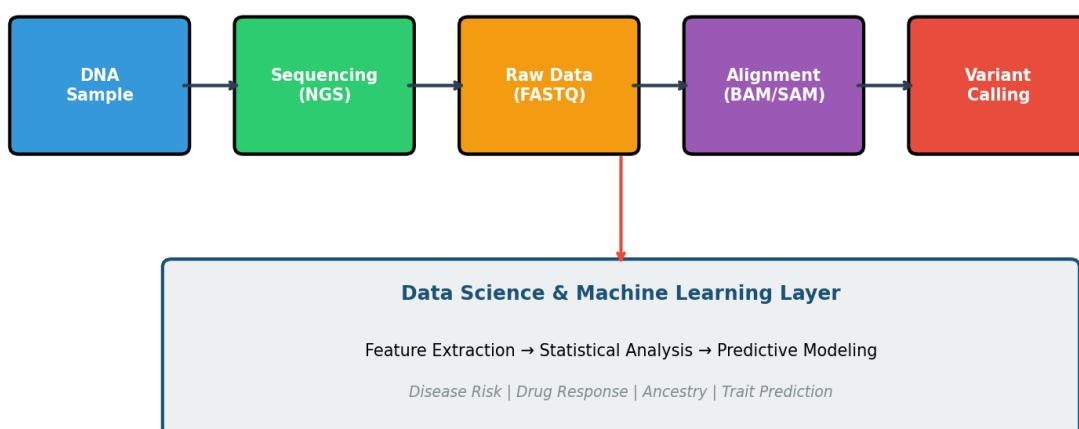
3.4 Massive Raw Data in Genomics

Next-Generation Sequencing (NGS) produces massive datasets requiring specialized algorithms for storage, alignment, and analysis.

Data Formats:

Format	Description	Size
FASTQ	Raw sequencing reads + quality scores	~100GB
SAM/BAM	Sequence Alignment Map (text/binary)	~50GB
VCF	Variant Call Format	~1GB
BED	Genomic intervals/features	~MB

Personal Genomics Data Pipeline



3.5 Data Science on Personal Genomes

Data science techniques enable extraction of meaningful insights from genomic data for personalized medicine and health prediction.

Applications:

- **Disease Risk Prediction:** GWAS identifies genetic variants associated with diseases
- **Pharmacogenomics:** Predict drug response based on genetic variants
- **Ancestry Analysis:** Population genetics and genealogy
- **Cancer Genomics:** Identify mutations driving tumor growth

Genome-Wide Association Study (GWAS):

Statistical test comparing allele frequencies between cases and controls:

$$\chi^2 = \sum (\text{Observed} - \text{Expected})^2 / \text{Expected}$$

P-value threshold: $p < 5 \times 10^{-8}$ (genome-wide significance)

3.6 Interconnectedness in Personal Genomics

Genomic data connects to broader biological networks including gene-gene interactions, gene-environment effects, and population-level patterns.

Network Analysis in Genomics:

- **Gene Regulatory Networks:** How genes regulate each other
- **Protein-Protein Interaction Networks:** Physical interactions between proteins
- **Metabolic Networks:** Biochemical pathways and reactions

UNIT IV: Machine Learning Fundamentals

4.1 Introduction to Machine Learning

Machine Learning is the study of algorithms that improve automatically through experience. It focuses on developing programs that access data and learn from it.

ML Pipeline:

1. Data Collection → 2. Data Preprocessing → 3. Feature Engineering
4. Model Selection → 5. Training → 6. Evaluation → 7. Deployment

4.2 Classification

Classification assigns input instances to discrete categories based on learned patterns. It's a supervised learning task with categorical output.

Types of Classification:

- **Binary:** Two classes (spam/not spam, positive/negative)
- **Multi-class:** Multiple mutually exclusive classes
- **Multi-label:** Multiple classes can be assigned simultaneously

Evaluation Metrics:

Metric	Formula	Use Case
Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	Balanced classes
Precision	$TP / (TP + FP)$	Minimize false positives
Recall	$TP / (TP + FN)$	Minimize false negatives
F1-Score	$2 \times (Precision \times Recall) / (P + R)$	Balance P and R
AUC-ROC	Area under ROC curve	Overall performance

4.3 Linear Classification

Linear classifiers separate classes using a linear decision boundary (hyperplane) in feature space.

4.3.1 Logistic Regression

Despite the name, logistic regression is a classification algorithm using the sigmoid function to map linear combinations to probabilities.

$$\text{Hypothesis: } h(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b) = 1 / (1 + e^{-(\mathbf{w}^T \mathbf{x} + b)})$$

Decision: Class 1 if $h(\mathbf{x}) \geq 0.5$, else Class 0

Loss Function (Binary Cross-Entropy):

$$L = -[y \log(h(\mathbf{x})) + (1-y) \log(1-h(\mathbf{x}))]$$

4.3.2 Support Vector Machine (SVM)

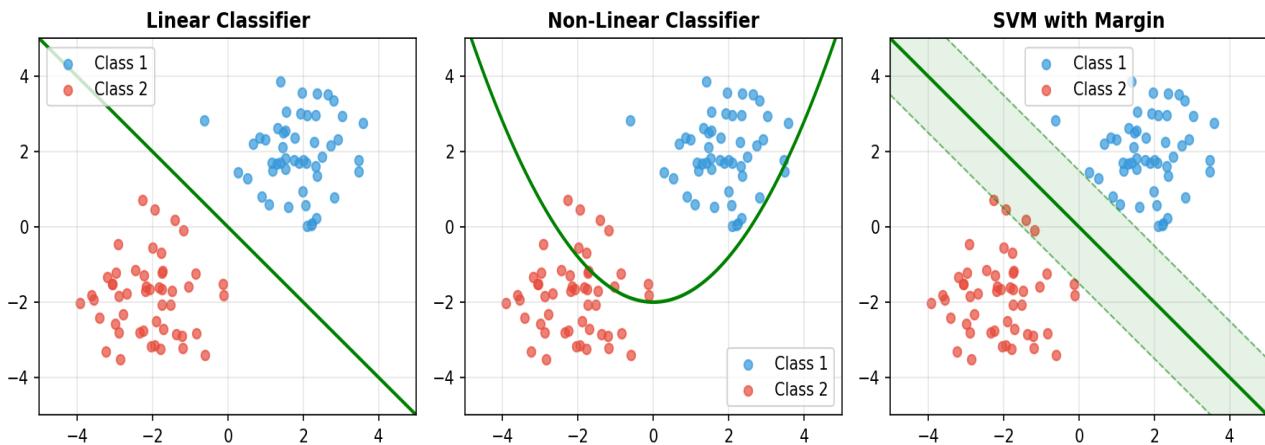
SVM finds the hyperplane maximizing margin between classes. Support vectors are points closest to the decision boundary.

Objective: Maximize margin = $2/\|w\|$

Decision Function: $f(x) = \text{sign}(w^T x + b)$

Kernel Trick: $K(x_i, x_j)$ maps to higher dimensions for non-linear separation

Classification Algorithms Comparison



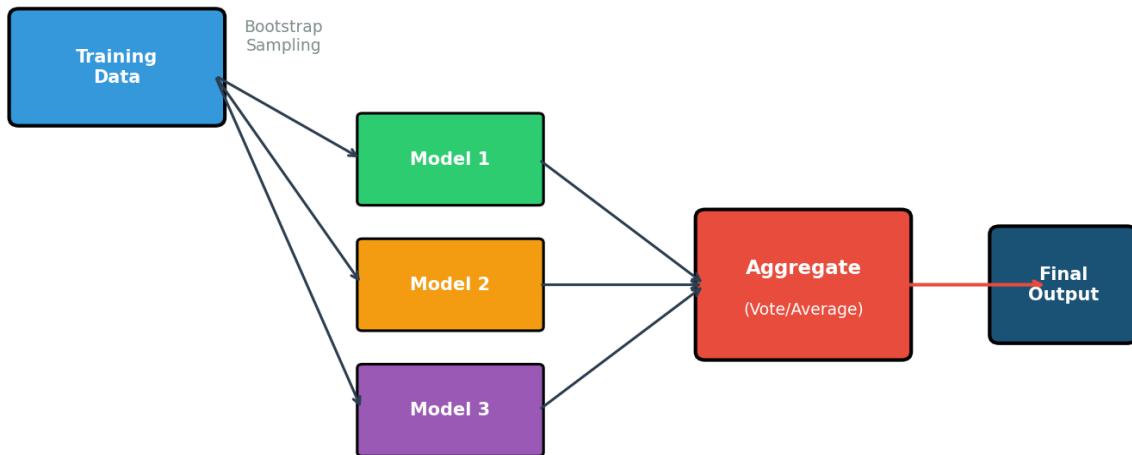
Common Kernels:

- **Linear:** $K(x,y) = x^T y$
- **Polynomial:** $K(x,y) = (x^T y + c)^n$
- **RBF (Gaussian):** $K(x,y) = \exp(-\gamma \|x-y\|^2)$

4.4 Ensemble Classifiers

Ensemble methods combine multiple models to achieve better performance than any single model. They reduce variance and/or bias.

Ensemble Methods: Bagging & Boosting



Bagging: Bootstrap samples → Train models → Average/Vote

Boosting: Sequential training → Weight misclassified → Combine

4.4.1 Bagging (Bootstrap Aggregating)

Train multiple models on bootstrap samples (random samples with replacement), then average predictions (regression) or vote (classification).

Example: Random Forest = Bagging + Feature Randomization + Decision Trees

4.4.2 Boosting

Sequential training where each model focuses on errors of previous models. Examples: AdaBoost, Gradient Boosting, XGBoost.

AdaBoost Weight Update: $w_i^{t+1} = w_i^t \times \exp(\alpha \times I(y_i \neq h_t(x_i)))$

Method	Reduces	Training	Example
Bagging	Variance	Parallel	Random Forest
Boosting	Bias	Sequential	AdaBoost, XGBoost
Stacking	Both	Meta-learning	Blending models

4.5 Model Selection

Model selection involves choosing the best model architecture and hyperparameters while avoiding overfitting.

Bias-Variance Tradeoff:

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- **High Bias:** Underfitting, model too simple
- **High Variance:** Overfitting, model too complex

Regularization:

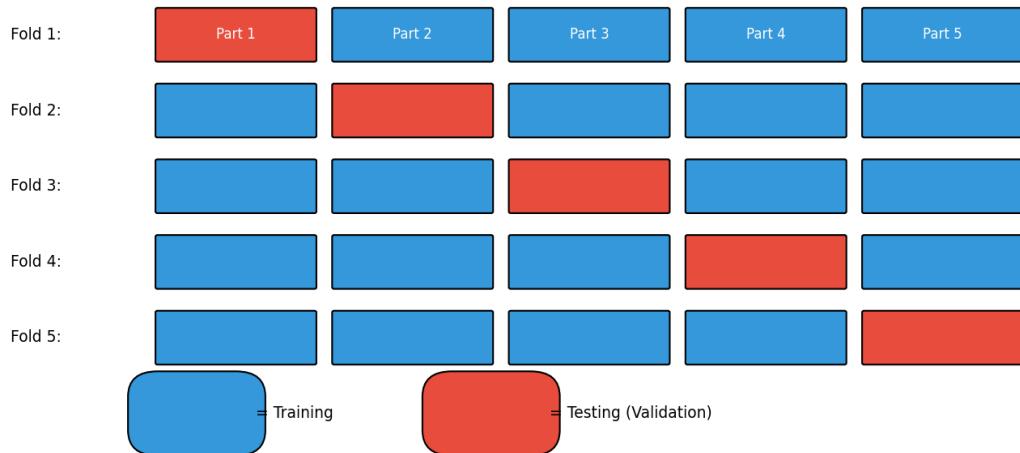
- **L1 (Lasso):** Loss + $\lambda \sum |w_i|$ → Sparse solutions

- **L2 (Ridge):** Loss + $\lambda \sum w^2$ → Small weights
- **Elastic Net:** Combination of L1 and L2

4.6 Cross-Validation

Cross-validation estimates model performance on unseen data by partitioning training data into complementary subsets for training and validation.

5-Fold Cross Validation



K-Fold Cross-Validation:

1. Split data into K equal folds
2. For each fold i: train on K-1 folds, test on fold i
3. Average performance across all K iterations

$$\text{Final Score} = (1/K) \sum \text{Score}_i \quad | \quad \text{Common: } K = 5 \text{ or } 10$$

Variants:

- **Stratified K-Fold:** Preserve class distribution in each fold
- **Leave-One-Out (LOOCV):** K = n, computationally expensive
- **Repeated K-Fold:** Multiple repetitions with different splits

4.7 Holdout Method

The simplest validation approach: split data into training and test sets once.

Common Splits:

- Training: 70-80% | Test: 20-30%
- Training: 60% | Validation: 20% | Test: 20%

Limitations:

- Results depend on particular split
- Less training data available

- High variance in performance estimates

UNIT V: Machine Learning Applications

5.1 Probabilistic Modeling

Probabilistic models express uncertainty about outcomes using probability distributions. They provide principled ways to handle noise, missing data, and make predictions with confidence.

Bayes' Theorem:

$$P(\theta|D) = P(D|\theta) \times P(\theta) / P(D)$$

$$\text{Posterior} = (\text{Likelihood} \times \text{Prior}) / \text{Evidence}$$

Key Concepts:

- **Prior $P(\theta)$:** Belief about parameters before seeing data
- **Likelihood $P(D|\theta)$:** Probability of data given parameters
- **Posterior $P(\theta|D)$:** Updated belief after seeing data

Naive Bayes Classifier:

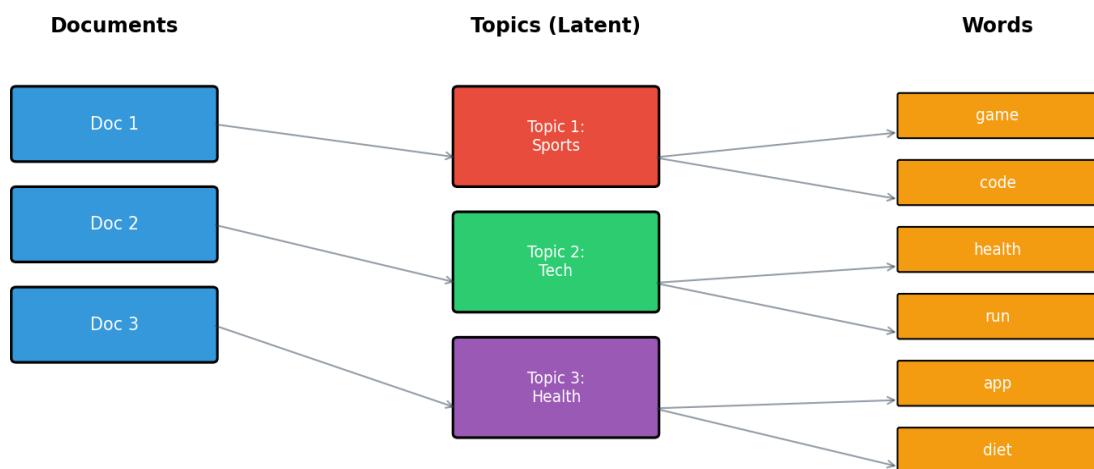
Assumes feature independence given class label. Despite this strong assumption, works well for text classification.

$$P(C|x_1, \dots, x_n) \propto P(C) \times \prod P(x_i|C)$$

5.2 Topic Modeling

Topic modeling discovers abstract topics in document collections. Documents are mixtures of topics, and topics are distributions over words.

Topic Modeling (Latent Dirichlet Allocation)



LDA: $P(\text{word}|\text{topic}) \times P(\text{topic}|\text{document}) \rightarrow \text{Document-Topic Distribution}$

Latent Dirichlet Allocation (LDA):

Generative Process:

1. For each document d : draw topic distribution $\theta_d \sim \text{Dirichlet}(\alpha)$
2. For each topic k : draw word distribution $\phi_k \sim \text{Dirichlet}(\beta)$
3. For each word position: draw topic $z \sim \text{Multinomial}(\theta_d)$
4. Draw word $w \sim \text{Multinomial}(\phi_z)$

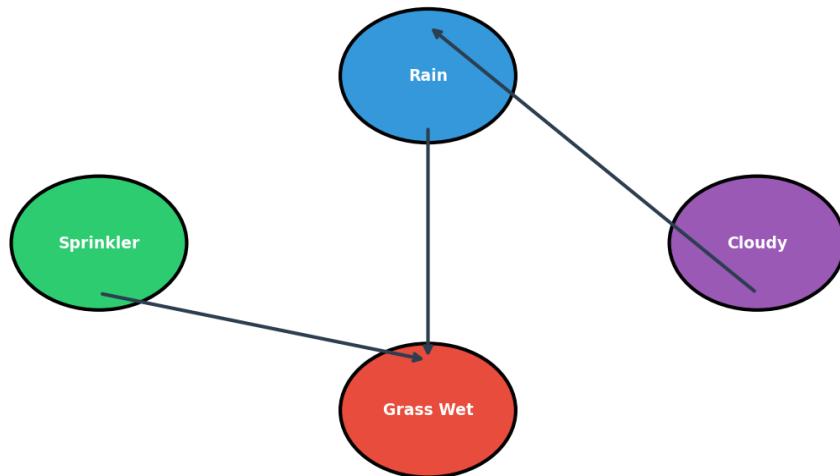
Parameters:

- α : Document-topic density (low = few topics per doc)
- β : Topic-word density (low = few words per topic)
- K : Number of topics (hyperparameter)

5.3 Probabilistic Inference

Probabilistic inference computes posterior distributions or expectations given observed evidence in probabilistic graphical models.

Bayesian Network for Probabilistic Inference



$P(\text{Grass Wet} | \text{Rain}, \text{Sprinkler})$ - Conditional Probability Table

Inference: Given evidence, compute $P(\text{Rain} | \text{Grass Wet} = \text{True})$

Inference Methods:

- **Exact Inference:** Variable elimination, junction tree algorithm
- **Approximate Inference:** Monte Carlo sampling, variational methods

Markov Chain Monte Carlo (MCMC):

Sampling method that constructs Markov chain whose stationary distribution is the target posterior.

Gibbs Sampling: Sample each variable conditioned on current values of others

Metropolis-Hastings: Accept/reject proposals based on acceptance ratio

5.4 Application: Prediction of Preterm Birth

Preterm birth prediction uses ML to identify high-risk pregnancies from clinical and biological data, enabling early intervention.

Problem Formulation:

- Binary classification: Preterm (<37 weeks) vs Full-term
- Imbalanced classes: ~10% preterm births
- Need high recall to not miss cases

Features Used:

Category	Examples
Demographics	Age, BMI, ethnicity, socioeconomic status
Medical History	Previous preterm births, cervical length, chronic conditions
Lab Results	Blood markers, hormone levels, infection indicators
Genomics	SNPs, gene expression, microbiome composition

Modeling Approaches:

- Logistic Regression with regularization
- Random Forest for feature importance
- Gradient Boosting (XGBoost) for best performance
- Deep learning for multi-modal data fusion

5.5 Data Description and Preparation

Data preparation transforms raw data into suitable format for ML algorithms. Quality of data preparation directly impacts model performance.

Data Preprocessing Steps:

1. **Data Cleaning:** Handle missing values, remove duplicates, fix errors
2. **Feature Scaling:** Standardization (z-score) or Normalization (min-max)
3. **Encoding:** One-hot encoding for categorical variables
4. **Feature Selection:** Remove irrelevant or redundant features
5. **Handling Imbalance:** SMOTE, undersampling, class weights

Scaling Formulas:

$$\text{Standardization: } z = (x - \mu) / \sigma$$

$$\text{Min-Max Normalization: } x' = (x - \text{min}) / (\text{max} - \text{min})$$

Missing Value Strategies:

Strategy	When to Use
Drop rows	Small % missing, MCAR (Missing Completely at Random)
Mean/Median imputation	Numerical features, missing not informative
Mode imputation	Categorical features
KNN imputation	Features correlated with neighbors

Model-based

Complex patterns, important features

5.6 Relationship Between ML and Statistics

Machine Learning and Statistics share foundations but differ in emphasis: ML focuses on prediction, Statistics on inference and understanding.



Both: Data-driven | Probabilistic | Model-based

Key Differences:

Aspect	Statistics	Machine Learning
Goal	Inference, hypothesis testing	Prediction accuracy
Model	Interpretable, parametric	Complex, flexible
Data Size	Small to medium	Large scale
Uncertainty	Confidence intervals, p-values	Cross-validation error
Features	Carefully selected	Many, automated selection

Convergence of Fields:

- Statistical learning theory provides foundations for ML
- Bayesian methods bridge both approaches
- Modern statistics adopts ML techniques for high-dimensional data
- Explainable AI brings interpretability back to ML

Quick Reference: Key Formulas

Topic	Formula
Big-O Definition	$f(n) = O(g(n)) \Leftrightarrow \exists c, n_0: 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0$
Master Theorem	$T(n) = aT(n/b) + f(n)$
Bayes Theorem	$P(A B) = P(B A) \cdot P(A) / P(B)$
Sigmoid	$\sigma(x) = 1 / (1 + e^{-x})$

Cross-Entropy	$H = -\sum y_i \log(p_i)$
F1 Score	$F1 = 2 \cdot (P \cdot R) / (P+R)$
Standardization	$z = (x - \mu) / \sigma$