Machine Learning in Bioinformatics: Comprehensive Study Material

Charles University Study Guide

June 2, 2026

Contents

1 Machine Learning in Bioinformatics - Introduction

1.1 Definitions of Bioinformatics

- "The use of computational methods to study biological data."
- "The application of computational methods to DNA and protein science."
- "The field of science in which biology, computer science, and information technology merge into a single discipline."
- "A multi-discipline, inter-discipline, and cross-discipline science for understanding biological systems, exploring underlying mechanisms of biological complexes, verifying biological hypotheses and providing evidence through in silico simulation for further theoretical development."

Determining the nucleotide sequence of a DNA molecule is only the first step towards the ultimate goals, which include understanding the functionality of DNA and genes, and knowing the locations of all the genes and regulatory sites of the molecule. Data analysis methods (knowledge discovery) are expected to support biologists in discovering patterns, understanding correlations, reducing complexity, and predicting events from massive datasets.

1.2 Machine Learning Concepts and Scenarios

Machine Learning is defined as computer science methods used to improve a performance criterion using example data or past experience.

Types of Machine Learning

- **Density estimation:** Learning the probability distribution according to which data has been sampled (from a pre-selected family of distributions or empirical).
- **Dimensionality reduction:** Finding a lower-dimensional manifold that preserves some properties of the data.
- **Clustering:** Partitioning data into homogenous groups.
- **Classification:** Assigning a category to each object.
- **Regression:** Predicting a real value for each object.
- **Ranking:** Ordering objects according to some criterion.

Terminology

- **Example (sample, observation, record):** An object or instance in the used data.
- **Features:** The set of attributes, often represented as a vector, associated to an example.
- **Labels:** In classification, the category associated to an object; in regression, real-valued numbers.
- **Training data:** Data used for training the algorithm.
- **Validation data:** Data used for tuning hyper-parameters of the training algorithm.
- **Test data:** Data exclusively used for testing the algorithm.

Learning Scenarios

- **Supervised learning:** Input relies on labeled training data with the goal to determine the labeling of new data. A finite set of labels implies classification, whereas an infinite set of labels (e.g., real numbers) implies regression.
- **Unsupervised learning:** Relies on data without labels with the goal to group similar data.
- **Semi-supervised learning:** Utilizes a small amount of labeled data with a large amount of unlabeled data (e.g., assuming points close to each other are more likely to share a label).

1.3 Algorithm: K-Fold Cross-Validation

1. Partition the data into K folds (typically 5 or 10).
2. Train the algorithm on all folds except the k-th fold to produce a hypothesis $h_{\theta,k}$, where $k \in [1, K]$.
3. Compute the fold cross-validation error using the formula:

$$\text{CV}_{\text{error}} = \frac{1}{K} \sum_{k=1}^{K} \text{error}(h_{\theta,k}, \text{fold } k) \tag{1}$$

   where $\text{error}(h_{\theta,k}, \text{fold } k)$ is the error of the specific hypothesis evaluated on fold k.
4. Compute the confidence interval for the error.
5. Choose the hyper-parameter value that minimizes the cross-validation error.

When the number of folds K equals the sample size m, this method is referred to as leave-one-out cross-validation.

1.4 Biology Minimum

Cellular Types:

- **Prokaryotes:** Single cell organisms lacking a nucleus and organelles, containing one piece of circular DNA and no mRNA post-transcriptional modification. Examples include Eubacteria and archaebacteria. The smallest cells known are bacteria; an E. coli cell contains roughly 3×10^6 protein molecules and 1000–2000 polypeptide species.
- **Eukaryotes:** Single or multi-cell organisms possessing a nucleus, organelles, chromosomes, and messenger ribonucleic acid. They undergo exons/introns splicing. Organisms include plants, animals, Protista, and fungi, and possess complex systems of internal membranes. A HeLa cell contains roughly 5×10^9 protein molecules and 5000–10,000 polypeptide species.

Genetics and Molecules of Life:

- **Genome:** An organism's complete set of DNA.
- **Gene:** Basic physical and functional units of heredity; specific sequences of DNA bases that encode instructions on how to make proteins. The mean gene size is 27 kbp.
- **Regulatory regions:** Extend up to 50 kb before a gene.

- **Exons:** Protein coding and untranslated regions (UTR). There are 1 to 178 exons per gene (mean 8.8), ranging from 8 bp to 17,000 bp per exon (mean 145 bp).
- **Introns:** Splice acceptor and donor sites, functioning as junk DNA. They average 1–50,000 bp per intron.
- **DNA (Deoxyribonucleic Acid):** Two complementary strands read in opposite directions holding cellular information, composed of 4 bases: Adenosine (A), Guanine (G), Cytosine (C), Thymidine (T).
- **RNA (Ribonucleic Acid):** Transfers short pieces of information to different parts of a cell and provides templates to synthesize proteins. Thymidine is replaced by Uracil (U).
- **Proteins:** Large, complex molecules made of smaller subunits called amino acids that make up cellular structure, form enzymes, and regulate gene activity. A protein's function is determined by its three-dimensional folded structure.

Central Dogma of Biology:
Information flows from DNA through transcription into hnRNA (heterogeneous nuclear RNA, pre-mRNA whose introns have not been excised), which is spliced into mRNA, exported from the nucleus, and translated at the ribosome into a protein.

- **Base Pairing Rule:** A and T (or U) are held together by 2 hydrogen bonds; G and C are held together by 3 hydrogen bonds.
- Translation occurs according to the genetic code, mapping successive triplets of RNA bases (codons) to amino acids.

Amino Acids List:
Alanine (Ala, A), Arginine (Arg, R), Asparagine (Asn, N), Aspartic acid (Asp, D), Cysteine (Cys, C), Glutamic acid (Glu, E), Glutamine (Gln, Q), Glycine (Gly, G), Histidine (His, H), Isoleucine (Ile, I), Leucine (Leu, L), Lysine (Lys, K), Methionine (Met, M), Phenylalanine (Phe, F), Proline (Pro, P), Serine (Ser, S), Threonine (Thr, T), Tryptophan (Trp, W), Tyrosine (Tyr, Y).

1.5 Reviewing Some Basic Statistics

Probability Distributions:
For a discrete random variable X with possible outcomes x_i, the probability mass function is given by:

$$p_X(x_i) = \Pr(X = x_i) \tag{2}$$

$$\sum_i p_X(x_i) = 1 \tag{3}$$

For a continuous random variable X, the probability density function (PDF) $f_X(x)$ satisfies:

$$\Pr[a \leq X \leq b] = \int_a^b f_X(x)dx \tag{4}$$

The cumulative density function (CDF) $F_X(x)$ is defined as:

$$F_X(x) = \int_{-\infty}^{x} f_X(u)du \tag{5}$$

$$\int_{-\infty}^{\infty} f_X(u)du = 1 \tag{6}$$

Expected Value and Sample Average:
The expected value (mean) $E[X]$ of a numeric random variable X is:

$$E[X] = \sum_x x \Pr(X = x) \quad \text{or} \quad \int_x x f_X(x) dx \tag{7}$$

Given N samples $x_1, \ldots, x_N$ from the same distribution, the sample average μ is an unbiased estimate of $E[X]$:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{8}$$

$$E\left[\frac{1}{N} \sum_{i=1}^{N} x_i\right] = E[X] \tag{9}$$

Variance:
The variance $\mathrm{Var}(X)$ is non-negative and captures the dispersion of the distribution:

$$\mathrm{Var}(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2 \tag{10}$$

The biased sample variance (maximum likelihood estimate) underestimates the true variance by a factor of $\frac{N-1}{N}$:

$$\sigma^2_{\text{biased}} = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2 \tag{11}$$

The unbiased estimator of $\mathrm{Var}(X)$ (unbiased sample variance) is:

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2 \tag{12}$$

1.6 Notation

- N: The number of observations.
- d: The number of variables or attributes.

A dataset matrix $\mathbf{X}$ is represented as an uppercase bold letter containing N observations and d features:

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \ldots & x_{Nd} \end{pmatrix} \tag{13}$$

A single observation x_i (1 row of $\mathbf{X}$) is denoted by a lowercase italicized letter and treated as a column vector by default:

$$x_i = (x_{i1}\ x_{i2}\ \ldots\ x_{id})^T = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{pmatrix} \tag{14}$$

The value of a single variable across all observations (a feature vector) $\mathbf{x}_j$ is denoted by a lowercase bold letter:

$$\mathbf{x}_j = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{Nj} \end{pmatrix} \tag{15}$$

The matrix $\mathbf{X}$ can therefore be structured as columns of features or rows of observations:

$$\mathbf{X} = (\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_d) = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix} \tag{16}$$

If there is a target variable y_i for the i-th observation, the observed data consists of pairs:

$$\{(x_1, y_1), \dots, (x_N, y_N)\} \tag{17}$$

Summary of Variable Typography:

- **A scalar** $a \in \mathbb{R}$: Lowercase, italics.
- **A vector of length** N $\mathbf{a} \in \mathbb{R}^N$: Lowercase, bold.

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix}$$

- **A vector of length** $k \neq N$ $a \in \mathbb{R}^k$: Lowercase, italics.
- **A matrix** $\mathbf{X} \in \mathbb{R}^{r \times s}$: Uppercase, bold.
- **A random variable** X: Uppercase, italics.

2 Unsupervised Learning in Bioinformatics

2.1 Probability Density Estimation Approaches

Probability density estimation attempts to find information and structures hidden within data by mapping it to a probability distribution. The data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are used to infer parameters and rules of an underlying probability structure.

2.2 Histogram Approach

The histogram approach is the simplest non-parametric method where explicit data structures are not assumed and training data is not kept.

- Each coordinate interval $[a, b]$ is divided into K segments of a fixed length called bins.
- The bin length Δ is defined as $\Delta = \frac{b-a}{K}$.
- The frequency of each bin is the ratio of training data falling into the bin relative to all training data.
- If n_i denotes the number of data points in bin i and N is the total number of points, the probability p_i of a point falling into bin i is calculated as $p_i = \frac{n_i}{N}$.
- The approximated probability density function f_i is computed as $f_i = \frac{n_i}{N\Delta}$.

This method suffers from discontinuities at bin boundaries and significant scaling issues. For d variables divided into K bins, the space requires K^d bins.

2.3 Parametric Approach

The parametric approach assumes the data follows a specific structure (e.g., normal distribution) prior to estimation. For a multidimensional Gaussian assumption:

- A training set of size N is represented as a matrix $\mathbf{X}$ where each column vector is $\mathbf{x}_i \in \mathbb{R}^d$.
- The goal is to maximize the likelihood function $\mathcal{L}$ that all N observations in $\mathbf{X}$ stem from the distribution $\mathcal{N}(\mu, \Sigma)$.

The likelihood function is defined as:

$$\mathcal{L} = \prod_{i=1}^{N} p(\mathbf{x}_i) \tag{18}$$

The multidimensional Gaussian probability density function for $\mathbf{x}_i$ is:

$$p(\mathbf{x}_i) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu)^T \Sigma^{-1} (\mathbf{x}_i - \mu)\right) \tag{19}$$

where $\Sigma \in \mathbb{R}^{d \times d}$ is the covariance matrix, Σ^{-1} is its inverse, $|\Sigma|$ is its determinant, and $\mu \in \mathbb{R}^d$ is the mean vector.

The sample mean vector μ and elements of the sample covariance matrix Σ_{ij} are defined as:

$$\mu = \frac{1}{N} \sum_{k=1}^{N} \mathbf{x}_k \tag{20}$$

$$\Sigma_{ij} = \frac{1}{N-1} \sum_{k=1}^{N} (x_{ki} - \mu_i)(x_{kj} - \mu_j) \tag{21}$$

If the training data attributes are orthogonal (independent), Σ becomes a diagonal matrix. Furthermore, if the data are orthogonal and homogenous ($\sigma_1 = \sigma_2 = \cdots = \sigma_d = \sigma$), the covariance matrix simplifies to $\Sigma = \sigma^2 \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

2.4 Non-Parametric Approach

Non-parametric approaches build models without a clearly defined structure, requiring all training data for predictions. Local density estimators approximate the density within a small region:

$$p(\mathbf{x}) = \frac{K}{NV} \tag{22}$$

where K is the number of points in the region, V is the volume of the region, and N is the total number of data points.

Kernel Density Estimators (KDE): Uses regions centered on data points that are allowed to overlap, avoiding discontinuities by utilizing regions with soft edges.

- Each region contributes $\frac{1}{N}$ to the total density, estimated as $p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\rho_i(\mathbf{x})$, where $\rho_i(\mathbf{x})$ acts as a kernel similarity function.
- A standard choice is the radial basis function (Gaussian kernel) parameterized by smoothing variable h:

$$\rho_i(\mathbf{x}) = \frac{1}{(2\pi h^2)^{d/2}} \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2h^2}\right)$$

K-Nearest Neighbor Approach: Similar to histograms, this method varies the size of a hyper-sphere around each test point such that exactly K training data points fall inside.

- The volume V^* is estimated from the data while K remains fixed, resulting in density $p(\mathbf{x}) = \frac{K}{NV^*}$.
- K acts as a smoothing parameter.

2.5 Semi-Parametric Approach

This method assumes that data is generated from a mixture model consisting of M Gaussians:

$$f(\mathbf{x}) = \sum_{m=1}^{M} w_m G_m(\mathbf{x}) \tag{23}$$

where $G_m(\mathbf{x})$ represents the m-th component Gaussian, and w_m represents the contribution of the m-th component subject to $0 \leq w_m \leq 1$ and $\sum_{m=1}^{M} w_m = 1$. Fitting this mixture model is typically executed using the Expectation-Maximization (EM) algorithm.

2.6 Dimensionality Reduction

Dimensionality reduction establishes a mapping ϕ of the set of data points $\mathcal{D} = \{\mathbf{x}_n \in \mathbb{R}^d\}_{n=1}^N$ to a new dimension $\tilde{d} \leq d$, forming $\tilde{\mathcal{D}} = \{\mathbf{y}_n \in \mathbb{R}^{\tilde{d}}\}_{n=1}^N$ such that $\phi(\mathbf{x}_n) = \mathbf{y}_n$. The mapping aims to preserve nearest-neighbor structures, i.e., minimizing $\|\mathbf{y}_m - \mathbf{y}_n\|$ proportionally.

Techniques divide into *Feature Selection* (choosing $k < d$ original features) and *Feature Extraction* (projecting into k derived dimensions).

Subset Selection Algorithms:

- *Forward search:* Iteratively adding the best feature x_j to a subset F if $\text{Error}(F \cup x_j) < \text{Error}(F)$. Uses hill-climbing to explore $O(d^2)$ subsets.
- *Backward search:* Starts with all features and removes the one causing the least error, step by step.
- *Floating search:* Alternates dynamically between adding k features to the selected set and removing l features from the possible pool.

2.7 Principal Components Analysis (PCA)

PCA finds a low-dimensional space projection minimizing information loss by maximizing the projected variance. Given a directional column vector $\mathbf{w}_1$, the length of the projection of $\mathbf{x}$ is $z = \mathbf{w}_1^T \mathbf{x}$.

To find $\mathbf{w}_1$, we maximize $\text{Var}(z)$ defined as:

$$\text{Var}(z) = E[(\mathbf{w}_1^T \mathbf{x} - E[\mathbf{w}_1^T \mathbf{x}])^2] = \mathbf{w}_1^T \Sigma \mathbf{w}_1 \tag{24}$$

To solve this subject to the constraint $\|\mathbf{w}_1\| = 1$, the Lagrange multiplier method introduces parameter $\alpha > 0$:

$$\max_{\mathbf{w}_1} \left(\mathbf{w}_1^T \Sigma \mathbf{w}_1 - \alpha(\mathbf{w}_1^T \mathbf{w}_1 - 1) \right) \tag{25}$$

Setting the derivative with respect to $\mathbf{w}_1$ to zero yields the eigenvector equation:

$$\Sigma \mathbf{w}_1 = \alpha \mathbf{w}_1 \tag{26}$$

Thus, $\mathbf{w}_1$ is selected as the eigenvector corresponding to the largest eigenvalue λ_1. The second principal component solves $\max \text{Var}(z_2)$ subject to $\|\mathbf{w}_2\| = 1$ and the orthogonality constraint $\mathbf{w}_2^T \mathbf{w}_1 = 0$.

The full PCA maps the data as $\mathbf{z} = \mathbf{W}^T(\mathbf{x} - \mu)$, where the columns of transformation matrix $\mathbf{W}$ are the sorted eigenvectors of Σ.

The number of principal components k is typically chosen using the Proportion of Variance (PoV) stopping condition (often PoV > 0.9) or via visual inspection of a scree graph at the "elbow":

$$\text{PoV} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{27}$$

2.8 Factor Analysis (FA)

FA seeks a small number of latent factors $z_1, \ldots, z_k \in \mathbb{R}^N$ $(k < d)$ which generate the observations via scaling and rotations:

$$\mathbf{x}_i - \mu_i = \sum_{j=1}^{k} v_{ij} z_j + \epsilon_i \tag{28}$$

where $\mathbf{V} = (v_{ij})_{d \times k}$ are the factor loadings, and ϵ_i represents noise sources. FA requires $E[z_j] = 0$, $\text{Var}(z_j) = 1$, and $\text{Cov}(z_i, z_j) = 0$ for $i \neq j$. Additionally, $\text{Cov}(\epsilon_i, \epsilon_j) = 0$ $(i \neq j)$ and $\text{Cov}(\epsilon_i, z_j) = 0$ for all variables. Comparatively, FA maps from $\mathbf{z}$ to $\mathbf{x}$, while PCA maps from $\mathbf{x}$ to $\mathbf{z}$.

2.9 Multidimensional Scaling (MDS)

MDS relies on placing items in a low-dimensional map to preserve pairwise distances given as $d_{i,j}^* = \|\mathbf{x}_i - \mathbf{x}_j\|$.

- *Classical MDS* applies linear transformation $\mathbf{z} = \mathbf{W}^T \mathbf{x}$.
- *General MDS* finds nonlinear parameterizations θ mapping $\mathbf{z} = g(\mathbf{x} \mid \theta)$ by minimizing the Sammon stress cost function:

$$E(\theta \mid \mathbf{X}) = \frac{1}{\sum_{i<j} d_{i,j}^*} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \frac{(d_{i,j}^* - \|\mathbf{z}_i - \mathbf{z}_j\|)^2}{d_{i,j}^*}$$

This metric is minimized iteratively using methods like gradient descent.

2.10 Linear Discriminant Analysis (LDA)

A supervised reduction mapping designed such that the projected representations $\mathbf{w}^T\mathbf{x}$ maximize the separation between pre-defined classes. LDA maximizes Fisher's linear discriminant $J(\mathbf{w})$:

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \tag{29}$$

where m_1, m_2 denote the projected centers of classes, and s_1^2, s_2^2 are their respective projected variances.

This corresponds to maximizing the between-class scatter $\mathbf{S}_B = (\bar{\mathbf{m}}_1 - \bar{\mathbf{m}}_2)(\bar{\mathbf{m}}_1 - \bar{\mathbf{m}}_2)^T$ while minimizing the within-class scatter $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}} \tag{30}$$

Solving for the derivative to be zero, the transformation vector results in $\mathbf{w} = \mathbf{S}_W^{-1}(\bar{\mathbf{m}}_1 - \bar{\mathbf{m}}_2)$.

For dimensionality reduction extending to $K > 2$ classes mapped onto dimension k via matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$, the scatter matrices adjust:

$$\mathbf{S}_W = \sum_{i=1}^{K} \sum_{t} r_t^{(i)} (\mathbf{x}_t - \mathbf{m}_i)(\mathbf{x}_t - \mathbf{m}_i)^T \tag{31}$$

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{32}$$

The optimal projection $\mathbf{W}$ comprises the largest eigenvectors derived from $\mathbf{S}_W^{-1}\mathbf{S}_B$.

2.11 Manifold-Based Extraction Algorithms

Isomap:
Replaces Euclidean distance with Geodesic distance to measure paths strictly along the underlying manifold.

- Constructs an adjacency graph where nodes r and s are connected with edge weight $\|\mathbf{x}_r - \mathbf{x}_s\|$ if they satisfy $\|\mathbf{x}_r - \mathbf{x}_s\| < \epsilon$ or if $\mathbf{x}_s$ falls within the k-nearest neighbors of $\mathbf{x}_r$.
- Distance for unconnected instances is calculated via the shortest path in the graph.
- The resulting $N \times N$ distance matrix is then passed into an MDS algorithm to extract the mapping.

Locally Linear Embedding (LLE):
Reconstructs each point from its neighbors. Follows a three-step algorithmic workflow:

- 1. For each $\mathbf{x}_r$, determine its S closest neighbors $\mathbf{x}_s^{(r)}$.
- 2. Identify the weight matrix $\mathbf{W}$ that minimizes the reconstruction error via least squares in the high-dimensional space:

$$E(\mathbf{W} \mid \mathbf{X}) = \sum_{r} \left\| \mathbf{x}_r - \sum_{s} \mathbf{W}_{rs} \mathbf{x}_s^{(r)} \right\|^2$$

  Subject to constraints $\mathbf{W}_{rr} = 0$ and $\sum_s \mathbf{W}_{rs} = 1$ across all r.

- 3. Hold the inferred $\mathbf{W}$ constant and solve for the lower dimensional representation vectors $\mathbf{z}_r$ that minimize the mapped error:

$$E(\mathbf{z} \mid \mathbf{W}) = \sum_r \left\| \mathbf{z}_r - \sum_s \mathbf{W}_{rs} \mathbf{z}_s^{(r)} \right\|^2$$

t-SNE (t-distributed Stochastic Neighbor Embedding):
Executes nonlinear transformations enforcing isolated parameters per region to retain local neighborhood distributions.

- 1. Assembles a probability distribution over original multi-dimensional point pairs, assigning higher probability metrics to highly similar instances and penalizing distant pairs.
- 2. Establishes a corresponding probability distribution within the reduced space framework and optimizes coordinate positions by minimizing the Kullback-Leibler divergence (KL divergence) between both distributions.
- 3. Important hyper-parameters dictate map consistency: *Perplexity* (a heuristic targeting the neighborhood density count), *Early exaggeration factor*, *Learning rate*, and *Maximum number of iterations*. Due to stochastic processes, multiple restarts with unique seeds will compute variable embedding topographies.

UMAP (Uniform Manifold Approximation and Projection):
Performs dimension reduction prioritizing balanced preservation of both global and local structures faster than t-SNE.

- 1. Builds a fuzzy, weighted high-dimensional graph. This acts by radiating a bounded radius outward from each vertex and drawing an edge if localized radii intersect. A small radius captures isolated clusters, while large radii construct complete graphs.
- To maintain local granularity, radii are dynamically defined by the distance to the k^{th} nearest neighbor (hyper-parameter `n_neighbors`), ensuring each point forms at least one definitive edge connection. Connection probabilities decrease outwards, making boundaries "fuzzy".
- 2. Employs a continuous force-directed graph layout optimizer to establish the most homologically similar configuration bound to the chosen lower dimension governed heavily by the tightness threshold parameter `min_dist`.

3 Machine Learning in Bioinformatics: Clustering

3.1 Density Estimation and Clustering Types

There are three primary types of methods for density estimation:

- **Parametric:** Assumes a single distinct model for $p(x \mid C_i)$.
- **Semiparametric:** Models $p(x \mid C_i)$ as a mixture of densities, which allows for multiple possible explanations or prototypes (e.g., different handwriting styles or speech accents).
- **Nonparametric:** Imposes no predefined model, allowing the data to speak for itself.

Clustering can be strictly considered as a type of density estimation. Grouping and partitioning data are two powerful approaches for discovering relevant biological regulations, which can subsequently be used in late hypothesis verification. In bioinformatics, clustering gene expression data reveals gene functions, signaling pathways, and cellular processes categories.

Types of clustering are broadly categorized into:

- **Grouping data:** Explores how data points are clustered through a learning process to reconstruct data relationships.
- **Partitioning data:** Discovers hidden data structure through a learning process.

3.2 K-Means Clustering Algorithm

The k-means clustering algorithm is a nonparametric method designed to find k reference vectors (also referred to as prototypes, codebook vectors, or codewords), denoted as m_j for $j = 1, \ldots, k$, which best represent the data.

For representing a given data point x_t, the algorithm selects the nearest (most similar) reference m_i such that the Euclidean norm is minimized:

$$\|x_t - m_i\| = \min_j \|x_t - m_j\| \tag{33}$$

The primary objective is to minimize the total reconstruction error (not the expected value) given the dataset X:

$$E(\{m_i\}_{i=1}^k \mid X) = \sum_{t=1}^{N} \sum_{i=1}^{k} b_t^{(i)} \|x_t - m_i\|^2 \tag{34}$$

Here, $b_t^{(i)}$ acts as a one-hot-encoding representing the encoder assignments, explicitly defined as:

$$b_t^{(i)} = \begin{cases} 1 & \text{if } \|x_t - m_i\| = \min_j \|x_t - m_j\| \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

Assuming the data lives in a Euclidean space and the target is k classes, the algorithm operates via the following workflow:

1. Initialize cluster centers m_i for $i = 1, \ldots, k$ (e.g., by selecting k random instances from X).
2. **Assignment step:** For all $x_t \in \{x_t\}_{t=1}^N$, assign each datapoint to the closest cluster by computing $b_t^{(i)}$ for all m_i.
3. **Refitting step:** Move each cluster center m_i to the center of gravity of the data currently assigned to it using the formula:

$$m_i = \frac{\sum_t x_t b_t^{(i)}}{\sum_t b_t^{(i)}} \tag{36}$$

4. Repeat the assignment and refitting steps until the centers m_i converge.

The algorithm is guaranteed to converge because whenever an assignment is changed, the sum squared distances of datapoints from their assigned centers is reduced. Similarly, whenever a cluster center is moved, the sum squared distances strictly reduce. The explicit test for convergence is met if the assignments do not change during the assignment step.

However, k-means can get stuck at local minima. To circumvent this, one could try multiple random starting points or implement non-local split-and-merge moves, which simultaneously split a large cluster into two or merge two nearby clusters.

The final cluster centers m_i heavily depend on their initial values. Common methods of initialization include:

- Selecting random k instances directly from the input.
- Computing the mean of the data and adding k small random vectors to the mean to serve as the initial $m_1, \ldots, m_k$.
- Computing the first principal component, dividing its range into k equal intervals, and partitioning the data into k groups based on those intervals.

3.3 Determining the Number of Clusters (k) and Evaluation Criteria

Choosing the correct number of centers k is often defined by the application (e.g., image quantization), by visually plotting data after PCA to check for clusters, or via an incremental leader-cluster algorithm that adds one cluster at a time until an "elbow" is observed in the reconstruction error.

Statistically, one can minimize information criteria to select the best model. Both criteria evaluate the general formula $-2\log\mathcal{L} + \lambda M$, where $\mathcal{L}$ is the model likelihood and M is the number of model parameters.

- **Akaike Information Criterion (AIC):** Sets $\lambda = 2$, yielding $-2\log\mathcal{L} + 2M$.
- **Schwarz's Bayesian Information Criterion (BIC):** Sets $\lambda = \log N$ (where N is the number of data points), yielding $-2\log\mathcal{L} + M\log N$.

For evaluation when the correct assignments C^* to the clusters are known, the following metrics are used:

- **Rand Index (RI):** Measures the similarity of predicted and true assignments. If a is the number of pairs assigned to the same cluster by both C and C^*, and b is the number of pairs assigned to different clusters by both, it is defined as:

$$\mathrm{RI} = \frac{a+b}{\binom{N}{2}}$$

- **Adjusted Rand Index (ARI):** Normalizes the RI so random clustering yields zero ARI. It is calculated as:

$$\mathrm{ARI} = \frac{\mathrm{RI} - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]}$$

When the correct assignments are not known, the Silhouette Coefficient s is utilized. For a single sample, let a be the mean distance between a sample and all other points in the same cluster, and let b be the mean distance between the sample and all other points in the next nearest cluster. The Silhouette Coefficient is defined as:

$$s = \frac{b-a}{\max(a,b)} \tag{37}$$

3.4 Fuzzy C-Means Clustering

Fuzzy c-means is a nonparametric clustering variation that allows instances (e.g., genes) to belong to more than one cluster. This allows a cluster to use more information about the data during the refitting step through soft assignments.

Instead of minimizing the hard reconstruction error, the algorithm incorporates a soft assignment $h_t^{(i)}$ of point x_t to cluster i, and minimizes the following objective for a constant $m \geq 1$:

$$\sum_{t=1}^{N}\sum_{i=1}^{k}(h_t^{(i)})^m\|x_t - m_i\|^2 \tag{38}$$

The cluster centers m_i and the soft assignments $h_t^{(i)}$ are formulated as:

$$m_i = \frac{\sum_t x_t(h_t^{(i)})^m}{\sum_t(h_t^{(i)})^m} \tag{39}$$

$$h_t^{(i)} = \frac{1}{\sum_{j=1}^{k}\left(\frac{\|x_t - m_i\|}{\|x_t - m_j\|}\right)^{\frac{2}{m-1}}} \tag{40}$$

Fuzzy c-means benefits from this continuous membership function, allowing the centers of clusters to be estimated more accurately. Note that "c" refers to the number of clusters, functioning identically to k in k-means.

3.5 Mixture Models and the EM Algorithm

Both k-means and fuzzy c-means algorithms are designed to detect cluster structure assuming homogenously distributed cluster densities in all dimensions (symmetrical volumes). Mixture models relax this assumption, providing a parametric model framework.

A mixture density is formally defined as:

$$p(x) = \sum_{i=1}^{k} p(x \mid G_i)P(G_i) \tag{41}$$

Where G_i represents the components/clusters, $P(G_i) \geq 0$ are the mixture proportions (priors) that satisfy $\sum_{i=1}^{k} P(G_i) = 1$, and $p(x \mid G_i)$ are the component densities. For a Gaussian mixture, $p(x \mid G_i) \sim \mathcal{N}(\mu_i, \Sigma_i)$ and the parameters $\Phi = \{P(G_i), \mu_i, \Sigma_i\}_{i=1}^{k}$ must be estimated from the unlabeled samples via unsupervised learning.

Density estimation maximizes the likelihood of the sample (incomplete log-likelihood) defined as:

$$\mathcal{L}(\Phi \mid X) = \log\prod_{t=1}^{N} p(x_t \mid \Phi) = \sum_t \log\sum_{i=1}^{k} p(x_t \mid G_i)P(G_i) \tag{42}$$

Because maximization cannot be solved analytically, the Expectation-Maximization (EM) algorithm assumes hidden variables Z representing the unobserved sources of observations. The algorithm iterates between two steps:

- **E-step:** Estimates Z given X and current parameters $\Phi^{(l)}$ using the expected complete log-likelihood:
$$\mathcal{Q}(\Phi \mid \Phi^{(l)}) = E[\mathcal{L}_c(\Phi \mid X, Z) \mid X, \Phi^{(l)}]$$
- **M-step:** Finds new parameters by maximizing $\mathcal{Q}$:
$$\Phi^{(l+1)} = \arg\max_{\Phi} \mathcal{Q}(\Phi \mid \Phi^{(l)})$$

An increase in $\mathcal{Q}$ strictly guarantees an increase in the incomplete log-likelihood: $\mathcal{L}(\Phi^{(l+1)} \mid X) \geq \mathcal{L}(\Phi^{(l)} \mid X)$.

In Gaussian Mixtures, we define an indicator variable $z_t^{(i)} = 1$ if x_t belongs to G_i, and 0 otherwise. In the E-step, the expected value of this variable (the soft label $h_t^{(i)}$) is calculated as:

$$\mathbb{E}[z_t^{(i)} \mid X, \Phi^{(l)}] = P(G_i \mid x_t, \Phi^{(l)}) = h_t^{(i)} = \frac{p(x_t \mid G_i, \Phi^{(l)})P(G_i)}{\sum_j p(x_t \mid G_j, \Phi^{(l)})P(G_j)} \tag{43}$$

For Gaussian components specifically, this expands to:

$$h_t^{(i)} = \frac{P(G_i)|\Sigma_i|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x_t - m_i)^T \Sigma_i^{-1}(x_t - m_i)\right]}{\sum_j P(G_j)|\Sigma_j|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x_t - m_j)^T \Sigma_j^{-1}(x_t - m_j)\right]} \tag{44}$$

During the M-step, Lagrangian multipliers are used to solve for the updated parameter estimates:

- **Component priors:** $P(G_i) = \frac{\sum_t h_t^{(i)}}{N}$
- **Component means:** $m_i^{(l+1)} = \frac{\sum_t h_t^{(i)} x_t}{\sum_t h_t^{(i)}}$
- **Covariance matrices:** $S_i^{(l+1)} = \frac{\sum_t h_t^{(i)} (x_t - m_i^{(l+1)})(x_t - m_i^{(l+1)})^T}{\sum_t h_t^{(i)}}$

The EM algorithm is typically initialized using k-means to estimate initial centers m_i, alongside computing S_i and initializing $P(G_i) = \frac{\sum_t b_t^{(i)}}{N}$. Notably, k-means clustering is a special case of EM applied to Gaussian mixtures when inputs are assumed independent with equal and shared variances, all components possess equal priors, and the resulting soft labels are hardened.

3.6 Mixtures of Latent Variable Models

Using full covariance matrices with Gaussian mixtures can trigger overfitting if the input dimensionality is high and the sample size is small. Regularizing clusters by assuming a strictly common or strictly diagonal covariance matrix is risky as it may improperly ignore cluster shapes or remove vital feature correlations.

To safely decrease dimensionality within clusters, one can use Mixtures of Latent Variable Models via PCA or Factor Analysis (FA). The modified component density is mathematically formulated as:

$$p(x_t \mid G_i) = \mathcal{N}(m_i, V_i V_i^T + \Psi_i) \tag{45}$$

Here, V_i represents the factor loadings of G_i, and Ψ_i accounts for the specific variance of G_i, with the parameter V_i being learnable via EM.

3.7 Hierarchical Clustering

Hierarchical clustering constructs nested groups based on deterministic similarities or distances. Two common distance measures between instances x_r and x_s include the Minkowski (L_p) distance and the City-block distance:

$$d_m(x_r, x_s) = \left[\sum_{j=1}^{d} (x_{rj} - x_{sj})^p\right]^{\frac{1}{p}} \tag{46}$$

$$d_{\text{cb}}(x_r, x_s) = \sum_{j=1}^{d} |x_{rj} - x_{sj}| \tag{47}$$

The agglomerative clustering algorithm takes a dataset X and a distance measure $d(G, G')$ to output a hierarchical tree (dendrogram). The steps are:

1. Put each x_t into its own distinct cluster $\{x_t\}$ on a working list W.
2. Repeat the following: Find two clusters G_i, G_j in W that possess the lowest distance according to $d(\cdot, \cdot)$. Remove both G_i and G_j from W, and insert their union $G_i \cup G_j$ back into W.
3. Terminate when W contains only a single cluster.

The distance measure between two groups G_i and G_j fundamentally dictates the shape of the dendrogram:

- **Single-link:** Computes the minimum distance $d(G_i, G_j) = \min_{x_r \in G_i, x_s \in G_j} d(x_r, x_s)$.
- **Complete-link:** Computes the maximum distance $d(G_i, G_j) = \max_{x_r \in G_i, x_s \in G_j} d(x_r, x_s)$.
- **Average-link:** Evaluates the average distance across all respective pairs.
- **Centroid distance:** Measures the distance strictly between the geometric centroids of the groups.

A measure satisfies monotonicity if, for any clusters A, B, and C, the conditions $d(A, B) < d(A, C)$ and $d(A, B) < d(B, C)$ imply:

$$d(A, B) < d(A \cup B, C) \tag{48}$$

Single-link, complete-link, and average-link satisfy monotonicity, ensuring that when two clusters merge, their inter-cluster distance is strictly not smaller than the distances corresponding to all prior merges. Centroid methods, often utilized in genomics, do not inherently satisfy monotonicity.

3.8 Density Based Clustering (DBSCAN)

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) relies on spatial densities to find core samples of high density and subsequently expand clusters outward from them.

The partition process is strictly driven by two arguments: `min_samples` and `eps`.

- Core samples are identified if there exist at least `min_samples` other points within the strict distance bounds of `eps`.
- These surrounding points are explicitly defined as the neighbors of the core sample.

The algorithm identifies a core sample, recursively aggregates all its neighbors that are also core samples into a unified cluster, and then appends non-core border samples that remain neighbors of a core sample. Any instances unable to connect to a dense core are universally classified as outliers.

3.9 After Clustering Analysis

Following the application of clustering algorithms, the derived structural boundaries are heavily leveraged for downstream feature engineering or supervised frameworks.

The original indicator variables (e.g., $h^{(j)}$ for soft distributions or $b^{(j)}$ for hard assignments) function directly as the dimensions of an entirely new k-dimensional feature space. This allows the training of a final discriminant or regressor upon this newly mapped, distributed representation.

4 Machine Learning in Bioinformatics: Classification

4.1 Supervised Learning Setting

A training dataset is formally defined as a set of pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i represents an object and y_i represents its corresponding class label. The element $x_i \in \mathbb{R}^d$ is conventionally termed the input vector, whereas $y_i \in \Theta$ is referred to as the target variable. In bioinformatics, elements of x are designated as explanatory variables that describe genotypic causes, while elements of y are predictive variables that describe observed phenotypic data. A mapping model is established to interpret phenotypic data such that the model acts as a function from explanatory variables to predictive variables. The dataset $\mathcal{D}$ is assumed to be randomly sampled from a space $\mathbb{R}^d \times \Theta$ that satisfies an unknown function mapping $f(x) \mapsto y$.

The domain of the target variable y, denoted by Θ, defines the specific learning problem:

- $\Theta = \{0, 1\}$ represents a binary classification problem.
- $\Theta = \{1, 2, \ldots, n\}$ denotes a multiclass classification problem.
- $\Theta = \mathbb{R}$ indicates a regression problem.

4.2 Classifier Overview

Several fundamental classification techniques are utilized:

- **Nearest Neighbor:** Classifies an object x' by identifying the most similar object x_i and assigning $y_i = y'$.
- **Naïve Bayes:** A probabilistic classifier applying Bayes' theorem alongside strong independence assumptions.
- **Decision Trees:** Constructs a hierarchy of attribute-based tests structured as a tree to classify an object.
- **Support Vector Machine (SVM):** Draws a hyperplane that separates two classes by maximizing the margin distance between the hyperplane and the closest data points.

4.3 Binary Classifier Evaluation Measures

Binary classifiers are primarily evaluated using the metrics of precision, specificity, sensitivity (recall), total accuracy, and the F-score. Precision is defined as:

$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{49}$$

Specificity is evaluated by:

$$\text{Spe} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{50}$$

Sensitivity is formulated as:

$$\text{Sen} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{51}$$

Total prediction accuracy is computed as:

$$\text{Tot} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{FP} + \text{TP} + \text{FN}} \tag{52}$$

The F-score (or F_1-score) combines precision and sensitivity:

$$F_1 = 2 \cdot \frac{\text{Prec} \cdot \text{Sen}}{\text{Prec} + \text{Sen}} \tag{53}$$

The Receiver Operating Characteristics (ROC) curve evaluates parametric two-class classifiers by plotting Sen on the vertical axis against $1 - \text{Spe}$ on the horizontal axis. The horizontal axis is expressed as:

$$1 - \text{Spe} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{54}$$

The overall criterion utilized for this curve is the Area Under the ROC Curve (AUC).

4.4 Nearest Neighbor Classifier

Given a new object x', its label y' is predicted by finding the nearest neighbor in the dataset. This neighbor is determined through the minimization of the squared distance:

$$x_i = \arg\min_{x \in \mathcal{D}} \|x - x'\|^2 \tag{55}$$

Consequently, the label is assigned as $y' = y_i$. The computational runtime requires $\mathcal{O}(N)$ operations for a single point and $\mathcal{O}(N^2)$ for an entire dataset. Performance and speed can be improved by pre-sorting training sets using kd-trees or Voronoi cells, or by utilizing Learning Vector Quantization (LVQ) to select a dataset subset.

4.5 Naïve Bayes Classifier

The Naïve Bayes method relies on Bayes' rule:

$$\Pr(C|x) = \frac{\Pr(x|C)\Pr(C)}{\Pr(x)} \tag{56}$$

To classify x' into one of K classes $(C_1, \ldots, C_K)$, the algorithm maximizes the probability:

$$\arg\max_{C_k} \Pr(C_k|x) = \frac{\Pr(x|C_k)\Pr(C_k)}{\Pr(x)} \tag{57}$$

Given that $\Pr(x)$ remains identical across all classes, it is omitted from the maximization.

$$\arg\max_{C_i} \Pr(C_i|x) \propto \Pr(x|C_i)\Pr(C_i) \tag{58}$$

For a multidimensional input vector $x = (x_1, \ldots, x_d)$, strong feature independence is assumed.

$$\Pr(x|C_k) = \Pr(C_k) \prod_{j=1}^{d} \Pr(x_j|C_k) \tag{59}$$

Applying this simplification, the core classification is evaluated proportionally:

$$\Pr(x|C_k) \propto \Pr(C_k) \prod_{j=1}^{d} \Pr(x_j|C_k) \tag{60}$$

In the Simple Bayes Classifier, an additional assumption states that $\Pr(C_k)$ is equal for all classes $1 \leq k \leq K$, allowing its removal.

$$\Pr(x|C_k) \propto \prod_{j=1}^{d} \Pr(x_j|C_k) \tag{61}$$

The overall time complexity is $\mathcal{O}(NKd)$.

4.6 Decision Trees and Information Metrics

Decision trees recursively partition the data space into pure regions. The model structure consists of a root node, internal nodes performing attribute tests, branches for outcomes, and leaf nodes containing class labels.

To construct the tree, an attribute selection measure such as Information Gain is utilized by algorithms like ID3. The information content (Shannon entropy) of a set $\mathcal{D}$ with elements distributed among classes $\{C_1, \ldots, C_K\}$ is defined as:

$$\text{Info}(\mathcal{D}) = -\sum_{k=1}^{K} p(C_k) \log_2 p(C_k) \tag{62}$$

The variable $p(C_k)$ represents the probability that an arbitrary tuple in $\mathcal{D}$ belongs to class C_k, estimated by class ratios in $\mathcal{D}$.

For a node m, if N_m instances reach it and $N_m^{(k)}$ of those belong to class C_k, the probability is estimated as:

$$\hat{\Pr}(C_k|x, m) = p_m^{(k)} = \frac{N_m^{(k)}}{N_m} \tag{63}$$

The node m is pure when $p_m^{(k)}$ equals 0 or 1. The impurity measure at node m is standard entropy:

$$J_m = -\sum_{k=1}^{K} p_m^{(k)} \log_2 p_m^{(k)} \tag{64}$$

For binary classification, this entropy minimizes at 0 (where $p = 0$ or 1) and maximizes at $p = 0.5$.

If node m is almost pure ($J_m < \Theta$), recursion stops. Otherwise, splitting creates branches. If N_{m_j} instances from N_m take branch j (where $1 \leq j \leq n$), and $N_{m_j}^{(k)}$ belong to C_k, the updated probability becomes:

$$\hat{\Pr}(C_k|x, m, j) = p_{m_j}^{(k)} = \frac{N_{m_j}^{(k)}}{N_{m_j}} \tag{65}$$

The impurity after the split is calculated as weighted entropy:

$$J'_m = \sum_{j=1}^{n} \frac{N_{m_j}}{N_m} \text{Info}(m_j) \tag{66}$$

The objective is to minimize this impurity across all valid attributes and split positions. Numeric attributes are evaluated on binary splits $X \leq v$ for $v \in [x_a, x_b)$, denoting distinct successive attribute values. Unordered categorical variables generally present $2^{d-1} - 1$ possible partitions, though they can be optimally handled by sorting occurrences in nodes for binary outcomes.

Alternatively, other impurity functions evaluate split validity:

- **Information Gain:** Maximized by ID3 as $J_m - J'_m$.
- **Normalized Information Gain:** Maximized by C4.5 as $(J_m - J'_m)/J'_m$.
- **Gini Index:** Defined for node D as $G(D) = 1 - \sum_{k=1}^{K} [p(C_k)]^2$, with post-split impurity measured as a weighted Gini index.

4.7 Regression Trees

In regression trees, leaves are denoted by an average target value g_m representing instances that reach node m. Let $b_m(x)$ act as an indicator function:

$$b_m(x) = \begin{cases} 1 & \text{if } x \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases} \tag{67}$$

The prediction value g_m is determined via:

$$g_m = \frac{\sum_t b_m(x_t) y_t}{\sum_t b_m(x_t)} \tag{68}$$

The squared error E_m evaluated at node m is:

$$E_m = \frac{1}{N_m} \sum_t (y_t - g_m)^2 b_m(x_t) \tag{69}$$

Post-split indicator behavior for branch j is written as:

$$b_{m_j}(x) = \begin{cases} 1 & \text{if } x \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases} \tag{70}$$

The newly computed leaf prediction g_{m_j} updates to:

$$g_{m_j} = \frac{\sum_t b_{m_j}(x_t) y_t}{\sum_t b_{m_j}(x_t)} \tag{71}$$

The resulting partitioned error E'_m becomes:

$$E'_m = \frac{1}{N_m} \sum_j \sum_t (y_t - g_{m_j})^2 b_{m_j}(x_t) \tag{72}$$

4.8 Decision Tree Generation Algorithms

Algorithms dynamically grow these trees using recursive logic.

Algorithm: GenerateTree(X)

- Calculate NodeEntropy(X).
- If NodeEntropy(X) $< \theta_I$, create a leaf labeled by the majority class present in X and return.
- Assign $i \leftarrow$ SplitAttribute(X).
- For each resulting branch of feature x_i:
- Find subset X_i falling directly into the current branch.
- Recursively call GenerateTree(X_i).

Algorithm: SplitAttribute(X)

- Initialize variable MinEnt $\leftarrow \infty$.
- Loop over all available attributes $i = 1, \ldots, d$.
- If x_i acts as a discrete attribute consisting of n distinct values:

- Split dataset X into partitions $X_1, \ldots, X_n$ based on x_i.
- Calculate continuous entropy $e \leftarrow \text{SplitEntropy}(X_1, \ldots, X_n)$.
- If $e <$ MinEnt, update MinEnt $\leftarrow e$ and best feature index bestf $\leftarrow i$.
- Else, implying x_i operates as a numeric attribute:
- Loop over all valid binary split points.
- Split dataset X into two partitions X_1, X_2 evaluated on x_i.
- Calculate binary entropy $e \leftarrow \text{SplitEntropy}(X_1, X_2)$.
- If $e <$ MinEnt, correspondingly update MinEnt $\leftarrow e$ and bestf $\leftarrow i$.
- Return identified optimal feature index bestf.

4.9 Tree Model Control and Instability

Decision trees frequently exhibit high variance (instability), indicating that minute changes in training data trigger heavily modified splits. To restrict tree size and counter overfitting, the Classification And Regression Tree (CART) algorithm leverages a grow-then-prune strategy. The algorithm generates an extensive tree and prunes it based on cross-validated criteria, which scales via the penalty formulation impurity $+ \alpha \cdot$ tree_size. For missing data, modern implementations formulate a discrete "missing" category or fall back on surrogate variables strictly available for split calculations.

5 Machine Learning in Bioinformatics: Support Vector Machines

5.1 The Perceptron

Given a training set of pairs $\{(x_i, y_i)\}_{i=1,\ldots,N}$ where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ for $i = 1, \ldots, N$ and some integer $d > 0$, the goal is to find a weight vector $\mathbf{w}$ and a constant threshold (or bias) b to correctly classify the data. The constraints are:

- $\mathbf{w}^T x_i + b \geq 0$ if $y_i = 1$
- $\mathbf{w}^T x_i + b < 0$ if $y_i = -1$

The perceptron learning algorithm proceeds through the following steps:

1. Initialize the weight vector $\mathbf{w}(0)$ and $b(0)$ with small random values, and set $t := 0$.
2. For $t = 1, \ldots, N$, compute the actual output of the perceptron $o = \text{sgn}(\mathbf{w}(t)^T x_t + b(t))$.
3. The sign function is defined as $\text{sgn}(x) = 1$ if $x \geq 0$, and -1 if $x < 0$.
4. Update weights based on the actual output:
   - If $o = y_i$, then $\mathbf{w}(t+1) = \mathbf{w}(t)$ and $b(t+1) = b(t)$.
   - Else if $o = -1$ and $y_i = 1$, then $\mathbf{w}(t+1) = \mathbf{w}(t) + x_t$ and $b(t+1) = b(t) + 1$.
   - Else if $o = 1$ and $y_i = -1$, then $\mathbf{w}(t+1) = \mathbf{w}(t) - x_t$ and $b(t+1) = b(t) - 1$.
5. Repeat step 2 until some stop condition is met.

Theorem: If the training set is linearly separable, the perceptron learning algorithm always finds a separating hyperplane such that all training samples will be correctly classified.

5.2 Classification Margin

The classification margin is defined as the maximal width of the symmetric band around the decision boundary that is completely devoid of any training samples. As the margin increases, the feasible region for the separating hyperplane reduces until only a single feasible separating hyperplane remains.

The margin ρ can be formalized in terms of the radius r of the largest empty circles (bubbles) centered on the boundary support vectors:

$$\rho = 2r \tag{73}$$

Support vectors are the sparse set of samples that directly touch the decision boundary; they completely control and define the boundary.

5.3 Vapnik-Chervonenkis (VC) Dimension and Loss Bounds

Let a dataset contain n points. These n points can be labeled in 2^n distinct ways as positive or negative, defining 2^n learning problems. If for any of these problems, a hypothesis $\gamma \in \mathcal{H}$ can successfully separate the positive from the negative examples with no error, then the hypothesis space $\mathcal{H}$ is said to shatter n points. The Vapnik-Chervonenkis (VC) dimension h is defined as the maximum number of points that can be shattered by $\mathcal{H}$. For the set of all hyperplanes in a d-dimensional space, $h = d + 1$.

The expected test risk $R(\alpha)$ for a selected decision boundary α is constrained by the training error $R_{\text{train}}(\alpha)$ and a monotonically increasing function of the VC-dimension h:

$$R(\alpha) \leq R_{\text{train}}(\alpha) + \sqrt{\frac{f(h)}{N}} \tag{74}$$

where $f(h) = h + h\log(2N) - h\log(h) - c$.

Minimizing the VC dimension directly improves generalization. The bound on the VC dimension based on the relative margin $\frac{\rho}{D}$ (where D is the data diameter) is:

$$h \leq \min\left(d, \left\lceil \frac{D^2}{\rho^2} \right\rceil\right) + 1 \tag{75}$$

Thus, maximizing the margin ρ minimizes the VC dimension h independently of the original dimensionality d.

5.4 Formalizing the Margin and the Optimization Problem

For a separable training set defined by a hyperplane $\mathbf{w}^T x + b = 0$, the geometric distance from the hyperplane to any support vector x_s is given by:

$$\frac{\rho}{2} = \frac{|\mathbf{w}^T x_s + b|}{\|\mathbf{w}\|} = \frac{y_s(\mathbf{w}^T x_s + b)}{\|\mathbf{w}\|} \tag{76}$$

Consequently, for all training vectors x_i, the constraint is:

$$y_i(\mathbf{w}^T x_i + b) \geq \frac{\rho}{2}\|\mathbf{w}\| \tag{77}$$

To strictly maximize the margin ρ, the right hand scaling is fixed such that $y_i(\mathbf{w}^T x_i + b) \geq 1$, which transforms the objective into minimizing the magnitude of $\mathbf{w}$.

The resulting quadratic optimization problem with linear constraints is constructed as:

$$\text{Minimize}: \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{78}$$

$$\text{Subject to}: \quad y_i(\mathbf{w}^T x_i + b) - 1 \geq 0, \quad \forall i \tag{79}$$

5.5 Lagrange Theory and Dual Form

By integrating the constraints, the primal Lagrangian form of the objective is established as:

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}^T x_i + b) - 1] \tag{80}$$

Subject to the multiplier constraints $\alpha_i \geq 0$ for all i.

Under Karush-Kuhn-Tucker (KKT) conditions, equating the gradients of the Lagrangian to zero yields necessary conditions for the optimum:

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} = 0 \implies \mathbf{w}^* = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{81}$$

$$\frac{\partial J(\mathbf{w}, b, \alpha)}{\partial b} = 0 \implies \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{82}$$

By complementary slackness, $\alpha_i[y_i(\mathbf{w}^{*T}x_i + b^*) - 1] = 0$, dictating that $\alpha_i > 0$ strictly if and only if x_i is a support vector.

Substituting these into J formulates the Dual Problem, which depends exclusively on dot products of the feature vectors:

$$\text{Maximize}: \quad D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j x_i^T x_j \tag{83}$$

Subject to the constraints:

- $\alpha_i \geq 0$
- $\sum_{i=1}^{N} \alpha_i y_i = 0$

Upon resolving the optimal α_i^*, the optimal boundary weights $\mathbf{w}^*$ and threshold b^* are derived:

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i \tag{84}$$

$$b^* = y_s - \mathbf{w}^{*T} x_s \tag{85}$$

where x_s is any support vector mapped to its label y_s.

5.6 Soft Margin Support Vector Machines (Handling Noise)

To accommodate non-separable classes and noisy instances, slack variables $\xi_i \geq 0$ are introduced to penalize misclassifications. The optimization objective is augmented to explicitly minimize training error weighted by a regularizer $C > 0$:

$$\text{Minimize}: \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N} \xi_i \tag{86}$$

$$\text{Subject to}: \quad y_i(\mathbf{w}^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \tag{87}$$

The primal Lagrangian incorporates the slack terms and a second set of non-negative multipliers μ_i:

$$J(\mathbf{w}, b, \alpha, \mu) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i[y_i(\mathbf{w}^T x_i + b) - 1 + \xi_i] - \sum_{i=1}^{N} \mu_i \xi_i \tag{88}$$

Calculating the partial derivative with respect to ξ_i establishes the relationship $\alpha_i + \mu_i = C$. Because $\mu_i \geq 0$, this bounds the upper limit of the alpha parameters:

$$\frac{\partial J}{\partial \xi_i} = 0 \implies \alpha_i \leq C \tag{89}$$

The dual problem remains structurally identical to the separable scenario but enforces a strict box constraint on α_i:

$$\text{Maximize}: \quad D(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j x_i^T x_j \tag{90}$$

Subject to:

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

5.7 Non-Linear Boundaries and the Kernel Trick

Data points can be transitioned into a higher-dimensional space via a non-linear mapping function $\Phi(x)$ to facilitate a linear boundary. Since the underlying optimization relies solely on dot products of vectors, a kernel function $K(x, y)$ can compute the higher-dimensional dot products strictly in the original input space, achieving significant computational efficiency:

$$K(x, y) = \Phi(x) \cdot \Phi(y) \tag{91}$$

Testing and predicting new unobserved samples directly implements this kernel computation:

$$\text{Label}(x_{\text{test}}) = \text{sgn}\left(\sum_{i=1}^{N}(\alpha_i y_i K(x_i, x_{\text{test}})) + b^*\right) \tag{92}$$

For a simple quadratic kernel mapping $\mathbb{R}^2 \to \mathbb{R}^4$, the explicit mapping $\Phi(x)$ is defined as:

$$\Phi(x) = \Phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2 x_1 \\ x_2^2 \end{bmatrix} \tag{93}$$

Using the quadratic kernel function computes this operation exponentially faster: $(x \cdot y)^2 = (x_1 y_1 + x_2 y_2)^2$.

Similarly, for a cubic mapping $\mathbb{R}^3 \to \mathbb{R}^{10}$, the full vector output is:

$$\Phi(x) = \begin{bmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ x_1^2 x_2 \\ x_1 x_2^2 \\ x_1^2 x_3 \\ x_1 x_3^2 \\ x_2^2 x_3 \\ x_2 x_3^2 \\ x_1 x_2 x_3 \end{bmatrix} \tag{94}$$

Which simplifies algebraically to the kernel $K(x, y) = (x \cdot y)^3$.

A generic polynomial kernel maps vectors into a higher dimensional space containing all lower-order multiplicative dimension variants up to power p:

$$K_p(x, y) = (1 + x^T y)^p = 1 + x^T y + (x^T y)^2 + \cdots + (x^T y)^p \tag{95}$$

Other common kernel functions defined structurally include:

- **Linear:** $K(x, y) = x^T y$
- **Gaussian (Radial Basis Function):** $K(x, y) = \exp\left(-\frac{\|x-y\|^2}{2s^2}\right)$

A necessary and sufficient condition for any given function $K(x, y)$ to act as a valid kernel is that the Gram matrix $\mathbf{K}$ computed over any arbitrary sequence $\{x_i\}$ must be strictly positive semidefinite. The matrix is constructed as:

$$\mathbf{K} = \begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots & K(x_1, x_N) \\ K(x_2, x_1) & K(x_2, x_2) & \dots & K(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_N, x_1) & K(x_N, x_2) & \dots & K(x_N, x_N) \end{pmatrix} \tag{96}$$

Where it must invariably hold that $x^T \mathbf{K} x \geq 0$.

5.8 Constructing Kernels

Given foundational valid kernels $K_1(x, y)$ and $K_2(x, y)$, new composite kernels may be correctly derived through standardized mathematical transformations ($c > 0$ is a scalar constant, $f(\cdot)$ is any scalar function, $q(\cdot)$ is a polynomial with non-negative coefficients, $\varphi(x)$ defines a mapping into $\mathbb{R}^M$, K_3 acts as a valid kernel inside $\mathbb{R}^M$, $\mathbf{A}$ establishes a symmetric positive semidefinite matrix, and input variables factor out as $x = (x_a, x_b)$):

- $K(x, y) = cK_1(x, y)$
- $K(x, y) = K_1(x, y) + K_2(x, y)$
- $K(x, y) = K_1(x, y)K_2(x, y)$
- $K(x, y) = f(x)K_1(x, y)f(y)$
- $K(x, y) = q(K_1(x, y))$
- $K(x, y) = \exp(K_1(x, y))$
- $K(x, y) = K_3(\varphi(x), \varphi(y))$
- $K(x, y) = x^T \mathbf{A} y$
- $K(x, y) = k_a(x_a, y_a) + k_b(x_b, y_b)$
- $K(x, y) = k_a(x_a, y_a)k_b(x_b, y_b)$

5.9 Solution Methods and Time Complexity

Support Vector Machine learning rigorously converges to global optimization targets absent of local minima due strictly to the mathematical properties underpinning its convex objective mapping. Analytical optimization solutions to matrix derivations restrictively limit datasets given a worst-case computational ceiling bounded at $\mathcal{O}(N_s^3)$ dictated primarily by the inversion limits of the algorithmic Hessian array, where N_s equals strictly the defined support vector count. Large problem environments consequently depend directly on general-purpose continuous numerical packages or algorithmic divide-and-conquer systems specifically crafted for linearly constrained convex programming.

The formal time complexity applied during runtime testing phases corresponds sequentially to:

$$\mathcal{O}(MN_s) \tag{97}$$

Where M equates distinctly to the quantity of independent operations essential to executing a localized kernel evaluation and N_s refers exactly to the isolated support vectors evaluated; in RBF bounds, M evaluates natively in $\mathcal{O}(d)$ bounds.

5.10 Applications in Bioinformatics

Class-boundary derivations strictly utilize one-versus-all algorithms spanning multi-class problems. Fundamental applications isolated across molecular biology strictly incorporate:

- Cancer Classification operating directly atop comparative Gene Expression Microarray datasets
- Protein Mutation Stability implementations computing localized structural variances
- Secondary Structure topology derivations

- Protein Fold mapping
- Protein Contact Map spatial matrices
- Universal Protein Structure Classification pipelines

Biological diagnostic mechanisms inherently mandate complex statistical mappings, including direct unsupervised learning isolated to recognizing independent tumor groups, supervised classification structuring defined malignancies, and discrete variable selection mechanisms highlighting individual isolated characteristic "marker" gene variables.

6 Ensemble Classification Methods: Bagging, Boosting, and Random Forests

6.1 Ensemble Methods Overview

Ensemble methods predict a class label for unseen data by aggregating a set of predictions from classifiers learned from the training data. The core idea is to build different "experts" and let them vote.

- **Advantages**: Improved predictive performance, direct inclusion of other types of classifiers, ease of implementation, and minimal parameter tuning.
- **Disadvantages**: The combined classifier is often a black box and lacks a compact representation.

6.2 Bagging (Bootstrap Aggregating)

Bagging reduces variance by voting or averaging across multiple models. It operates as follows:

- **Training**: Given a dataset S, at each iteration i, a training set S_i is sampled with replacement (bootstrapping) from S. A classifier C_i is learned for each S_i.
- **Classification**: For an unseen sample X, each classifier C_i returns its class prediction. The bagged classifier H counts the votes and assigns the class with the most votes to X.
- **Regression**: Continuous values are predicted by taking the average value of each individual prediction.

Bagging is most effective when the underlying learning algorithm is unstable, meaning that small changes to the training set cause large changes in the learned classifier (e.g., decision trees, regression trees, linear regression, and Support Vector Machines).

Mathematical Setup of Bootstrapping: Given a dataset of size N, a bootstrapped replicate is extracted with replacement. The fraction of never-selected instances can be estimated as:

$$\left(1 - \frac{1}{N}\right)^N \approx \frac{1}{e} \approx 0.368 \tag{98}$$

For T bootstrapped training sets, the number of samples never selected asymptotically approaches zero:

$$\left(1 - \frac{1}{N}\right)^{NT} \xrightarrow{T\to\infty} 0 \tag{99}$$

Proof of Convergence for Bagging: Let $S = \{(x_i, y_i), i = 1, \ldots, N\}$ be the set of training samples. We generate $S^{(1)}, \ldots, S^{(m)}$ via bootstrapping and compute continuous predictions $Y^{(1)}, \ldots, Y^{(m)}$. The aggregated prediction Z is:

$$Z = \frac{1}{m} \sum_{i=1}^{m} Y^{(i)} \tag{100}$$

Assuming $Y^{(1)}, \ldots, Y^{(m)}$ are independent and identically distributed, and $E[Y] = y$ makes it an unbiased estimator, the expected error variance is calculated as:

$$E[(Z - y)^2] = E\left[\left(\frac{1}{m} \sum_{i=1}^{m} Y^{(i)} - y\right)^2\right] = \frac{1}{m^2} \sum_{i=1}^{m} \sigma^2(Y^{(i)}) = \frac{1}{m}\sigma^2(Y) \tag{101}$$

Thus, the variance shrinks with m. If the unbiased assumption is dropped, the error decomposes as:

$$E[(Y-y)^2] = E[(Y-E[Y])^2] + E[(E[Y]-y)^2] + E[2(Y-E[Y])(E[Y]-y)] \tag{102}$$

Because $E[Y-E[Y]]=0$, the cross term vanishes, resulting in:

$$E[(Y-y)^2] \geq E[(E[Y]-y)^2] \tag{103}$$

This indicates that the expected error of the individual model bounds the aggregated error $E[(Y-y)^2] \geq E[(Z-y)^2]$.

6.3 Adaptive Boosting (AdaBoost)

Boosting builds models sequentially, where each learner aims to complement the previous ones to avoid redundancy.

Weak Learner Definition: A learning algorithm $\mathcal{A}$ is a γ-weak-learner for a class of hypotheses $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for every $\delta \in (0,1)$, for every probability distribution D over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{-1,+1\}$, running the algorithm on $m > m_{\mathcal{H}}(\delta)$ i.i.d. examples generated by D returns a hypothesis h. With probability of at least $1-\delta$, the error of h is at most $\frac{1}{2} - \gamma$.

Decision Stumps as Weak Learners: Let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{H}_{DS}$ be the set of decision stumps:

$$\mathcal{H}_{DS} = \{x \to \text{sign}(x_j - \theta) \cdot b \mid \theta \in \mathbb{R}, j \in \{1,\dots,d\}, b \in \{\pm 1\}\} \tag{104}$$

The algorithm finds optimal parameters b^*, j^*, θ^* minimizing the weighted error:

$$\min_{b\in\{\pm1\}, j\in\{1,\dots,d\}} \min_{\theta\in\mathbb{R}} \left(\sum_{i:y_i=1} D_i \mathbf{1}_{[h(x_{i,j})=-1]} + \sum_{i:y_i=-1} D_i \mathbf{1}_{[h(x_{i,j})=1]} \right) \tag{105}$$

Alternatively formulated as:

$$\min_{b\in\{\pm1\}, j\in\{1,\dots,d\}} \min_{\theta\in\mathbb{R}} \left(\sum_{i:y_i=1} D_i \frac{(1-h(x_i))}{2} + \sum_{i:y_i=-1} D_i \frac{(1+h(x_i))}{2} \right) \tag{106}$$

To achieve $\mathcal{O}(dN)$ time complexity, the samples are sorted along the j-th coordinate and thresholds are restricted to $\Theta_j = \left\{ \frac{x_{i,j}+x_{i+1,j}}{2} \mid i \in \{1,\dots,N\} \right\} \cup \{(x_{1,j}-1), (x_{N,j}+1)\}$.

AdaBoost Algorithm [Freund and Schapire, 1997]:

- **Input**: Training set $S = \{(x_1,y_1),\dots,(x_N,y_N)\}$, weak learner $\text{WL}(D,S)$, and number of rounds T.
- **Initialization**: Initialize uniform weights $D^{(1)} = (1/N,\dots,1/N)$.
- **Iteration**: For $t = 1,\dots,T$:
  - Invoke weak learner $h_t = \text{WL}(D^{(t)}, S)$.
  - Compute the weighted error ϵ_t:

$$\epsilon_t = \sum_{i=1}^{N} D_i^{(t)} \mathbf{1}_{[y_i \neq h_t(x_i)]}$$

  - Compute the ensemble weight α_t:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1}{\epsilon_t} - 1\right)$$

– Update the data distribution vector D:

$$D_i^{(t+1)} = \frac{D_i^{(t)} e^{-\alpha_t y_i h_t(x_i)}}{\sum_{j=1}^{N} D_j^{(t)} e^{-\alpha_t y_j h_t(x_j)}}$$

- **Output**: Formulate the final combined hypothesis:

$$h_S(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

AdaBoost Convergence Theorem: Assuming the weak learner returns a hypothesis with error $\epsilon_t < \frac{1}{2} - \gamma$ at each iteration, the training error of the combined output hypothesis h_S is upper bounded by:

$$L_S(h_S) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{[h_S(x_i) \neq y_i]} \leq e^{-2\gamma^2 T} \tag{107}$$

By defining $f_t = \sum_{p=1}^{t} \alpha_p h_p$, the distribution update effectively computes normalized exponential losses over the additive model:

$$D_i^{(t+1)} = \frac{e^{-y_i f_t(x_i)}}{\sum_{j=1}^{N} e^{-y_j f_t(x_j)}} \tag{108}$$

6.4 Random Forests

Random Forests construct multiple decision trees (e.g., CART) on bootstrapped dataset replicates and aggregate predictions via voting. The algorithm employs structural randomization:

- At each node, $m = \sqrt{p}$ candidate splitting variables are randomly sampled from the p available features.
- Cut-off values are evaluated across the m features, and the split minimizing the GINI impurity score is selected.
- The process recurses until node purity is achieved.

GINI Impurity Score: Given f_i as the fraction of items in a set belonging to category $i \in \{1, 2, \ldots, m\}$:

$$\text{Gini}(f) = \sum_{i=1}^{m} f_i(1 - f_i) = 1 - \sum_{i=1}^{m} f_i^2 \tag{109}$$

Additionally, standard evaluation metrics natively analyzed in random forest outputs are defined as:

$$\text{Sen} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{110}$$

$$\text{Spe} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{111}$$

6.5 Variable Importance (VI)

Variable Importance metrics evaluate the contribution of individual features:

- **Gini Impurity Score** (VI_{gini}): Defines importance as the average Gini impurity decrease across all trees at splits utilizing the variable.

- **Permutation Importance ($\mathrm{VI}_{\mathrm{perm}}$)**: Computes the mean decrease in out-of-bag (OOB) accuracy following feature permutation.

Permutation VI Estimation Algorithm:

- For each bootstrapped replicate, isolate the OOB instances.
- Evaluate the baseline predictions and cumulate correct counts.
- For every feature $j \in \{1, \ldots, d\}$:
  - Randomly permute values of x_j in the OOB sample.
  - Perform prediction using the permuted dataset.
  - Cumulate the degraded correct prediction counts.
- Importance is quantified as the average accuracy difference between original and permuted inputs.

6.6 Boruta Algorithm

Boruta uses random forests and shadow variables for robust feature selection:

- Introduce shadow variables as shuffled copies of all existing variables, ensuring a minimum of 5 shadow attributes.
- Train a Random Forest classifier on the extended attribute set and extract variable Z-scores.
- Determine the Maximum Z-Score among shadow attributes (MZSA).
- Execute a two-sided equality test against the MZSA for all unclassified attributes.
- Reject attributes exhibiting importance significantly lower than the MZSA.
- Retain attributes demonstrating importance significantly higher than the MZSA.
- Eliminate shadow features and recursively apply the procedure until attribute importances are conclusively resolved or an epoch limit is reached.

Z-Score Metric Formulation: The Z-Score evaluates mean classification accuracy loss scaled by standard deviation:

$$\text{Z-score} = \frac{\text{average\_loss}}{\text{std(average\_loss)}} \tag{112}$$

This score accounts for variance among internal trees but does not strictly map to a statistical significance threshold, as its distribution deviates from $\mathcal{N}(0, 1)$.

6.7 Conditional Variable Importance

To mitigate bias toward correlated predictors, conditional permutation executes shuffling of x_j strictly within subsets possessing equivalent values (or intervals) of correlated variables $Z = z$. This preserves the internal correlation structure among extraneous predictors, ensuring VI reflects strictly independent interactions with the target vector.

7 Machine Learning in Bioinformatics: Hidden Markov Models

7.1 Markov Models and CG-Islands

A time-homogeneous Markov model of order 1 is defined as a system $M = (Q, A)$ consisting of a finite set of states $Q = \{s_1, \ldots, s_k\}$ and a $|Q| \times |Q|$ transition matrix $A = (a_{kl})$. The matrix A represents the probability of changing from state s_k to state s_l, constrained such that for all $k \in Q$:

$$\Pr(x_{i+1} = s_l \mid x_i = s_k) = a_{kl} \quad \text{with} \quad \sum_{l \in Q} a_{kl} = 1 \tag{113}$$

A Markov chain is a sequence $x_0, x_1, \ldots, x_n$ of random variables in Q satisfying the Markov property:

$$\Pr(x_n = s \mid \bigcap_{j<n} x_j) = \Pr(x_n = s \mid x_{n-1}) \tag{114}$$

To model the beginning and end of a sequence explicitly, state sets often include a Begin state ($x_0 = \text{Begin}$) and an End state. The probability of stepping through a precise sequence of states $x = x_1, x_2, \ldots, x_L$ is given by:

$$\Pr(x) = \Pr(x_1) \prod_{i=2}^{L} a_{x_{i-1}, x_i} = \prod_{i=1}^{L} a_{x_{i-1}, x_i} \tag{115}$$

For a discrimination problem (e.g., classifying a sequence as a CG-island or not), transition matrices are computed from a training set. The transition probability from state s to t in a positive model is calculated as:

$$p^+_{st} = \frac{c^+_{st}}{\sum_{t'} c^+_{st'}} \tag{116}$$

Given a short segment x, we compute the log-odds ratio $S(x)$ to discriminate whether the sequence comes from a positive model (model$^+$) or a negative model (model$^-$):

$$S(x) = \log \frac{\Pr(x \mid \text{model}^+)}{\Pr(x \mid \text{model}^-)} = \sum_{i=1}^{L} \log \left(\frac{a^+_{x_{i-1}, x_i}}{a^-_{x_{i-1}, x_i}} \right) = \sum_{i=1}^{L} \beta_{x_{i-1}, x_i} \tag{117}$$

A Neyman-Pearson test uses a threshold c^* to classify the sequence:

$$\phi^*(x) = \begin{cases} 1 & \text{if } S(x) > c^* \\ 0 & \text{if } S(x) \leq c^* \end{cases} \tag{118}$$

7.2 Hidden Markov Models (HMM)

An HMM is represented by parameters $M(\Sigma, Q, A, E)$:

- Σ: A discrete alphabet of emission characters.
- Q: A finite set of hidden states.
- $A = (a_{kl})$: A $|Q| \times |Q|$ matrix of transition probabilities from state k to state l.
- $E = (e_k(b))$: A $|Q| \times |\Sigma|$ matrix of emission probabilities, representing the probability of emitting symbol b while in state k.

A hidden path $\pi = \pi_1 \ldots \pi_n$ is a sequence of states. The joint probability that the HMM follows the path π and emits the sequence $x = x_1 \ldots x_n$ is:

$$\Pr(x, \pi) = \Pr(x \mid \pi) \Pr(\pi) = \prod_{i=1}^{n} a_{\pi_{i-1}, \pi_i} e_{\pi_i}(x_i) \tag{119}$$

7.3 Decoding Problem and Viterbi Algorithm

The decoding problem seeks the path π that maximizes the joint probability $\Pr(x, \pi)$ for a given observation x. This is modeled as finding the longest path in a Directed Acyclic Graph (DAG) using dynamic programming.

Let $s_{l,i}$ be the logarithm of the probability of the most likely path generating the prefix $x_1, \ldots, x_i$ and ending in state l. The Viterbi algorithm workflow is as follows:

- **Initialization:**

$$s_{\text{Begin},0} = \log 1 = 0$$

$$s_{k,0} = \log 0 = -\infty \quad \text{for all } k \neq \text{Begin}$$

- **Iteration:** For $i = 1$ to n and for each state $l \in Q$:

$$s_{l,i} = \log e_l(x_i) + \max_{k \in Q} \{ s_{k,i-1} + \log a_{kl} \}$$

- **Termination and Output:** Backtrack from the maximum state score at step n to find the optimal path π^*:

$$\Pr(x, \pi^*) = \max_{k \in Q} s_{k,n}$$

7.4 Outcome Likelihood and Forward-Backward Algorithm

The Outcome Likelihood problem computes the total probability $\Pr(x)$ that the HMM emits sequence x over all possible paths.

The **Forward algorithm** computes $f_{k,i}$, the probability of emitting the prefix $x_1 \ldots x_i$ and reaching state $\pi_i = k$:

- **Initialization:** $f_{\text{Begin},0} = 1$ and $f_{k,0} = 0$ for all $k \neq$ Begin.
- **Iteration:**

$$f_{l,i} = e_l(x_i) \sum_{k \in Q} f_{k,i-1} a_{kl}$$

- **Termination:** $\Pr(x) = \sum_{k \in Q} f_{k,n} a_{k,\text{End}}$

The **Backward algorithm** computes $b_{k,i}$, the probability of emitting the suffix $x_{i+1} \ldots x_n$ given that the HMM is currently in state $\pi_i = k$:

- **Initialization:** $b_{k,n} = a_{k,\text{End}}$ for all $k \in Q$.
- **Iteration:**

$$b_{l,i} = \sum_{k \in Q} e_k(x_{i+1}) a_{lk} b_{k,i+1}$$

Using both, the posterior probability of being in state k at position i given the entire sequence x is:

$$\Pr(\pi_i = k \mid x) = \frac{\Pr(x, \pi_i = k)}{\Pr(x)} = \frac{f_{k,i} b_{k,i}}{\Pr(x)} \tag{120}$$

7.5 HMM Parameter Estimation

Parameter estimation seeks to find HMM parameters Θ that maximize the log-likelihood of independent training sequences $x^{(1)}, \ldots, x^{(m)}$:

$$\max \sum_i \log \Pr(x^{(i)} \mid \Theta) \tag{121}$$

Case 1: Known Paths If the paths are known, transitions (A_{kl}) and emissions ($E_k(b)$) are counted directly. To avoid zero probabilities (overfitting), pseudocounts r_{kl} and $r_k(b)$ are added:

$$a_{kl} = \frac{A_{kl} + r_{kl}}{\sum_{l'}(A_{kl'} + r_{kl'})} \quad \text{and} \quad e_k(b) = \frac{E_k(b) + r_k(b)}{\sum_{b'}(E_k(b') + r_k(b'))} \tag{122}$$

Case 2: Unknown Paths (Baum-Welch Algorithm) The Baum-Welch approach is an Expectation-Maximization (EM) algorithm that infers unobserved paths.

The expected number of times a transition $k \to l$ is used is defined as A_{kl}. It relies on the probability of traversing the edge from state k at i to state l at $i+1$:

$$\Pr(\pi_i = k, \pi_{i+1} = l \mid x) = \frac{b_{l,i+1} e_l(x_{i+1}) a_{kl} f_{k,i}}{\Pr(x)} \tag{123}$$

$$A_{kl} = \sum_x \sum_i \frac{b_{l,i+1} e_l(x_{i+1}) a_{kl} f_{k,i}}{\Pr(x)} \tag{124}$$

The expected number of times symbol b is emitted from state k is $E_k(b)$:

$$E_k(b) = \sum_x \sum_{i:x_i=b} \frac{f_{k,i} b_{k,i}}{\Pr(x)} \tag{125}$$

The new parameters are updated iteratively until the log-likelihood converges. To avoid numerical underflow during implementation, a sequence-independent scaling factor c_t is applied:

$$c_t = \frac{1}{\sum_{k=1}^{n} f_k(i)} \tag{126}$$

7.6 Sequence Alignment and Affine Gap Penalties

To align two sequences x and y of lengths m and n, standard global alignment with linear gaps uses a penalty $\gamma(g) = -gd$. The affine gap penalty model introduces a gap open penalty d and extension penalty e, such that $\gamma(g) = -d - (g-1)e$.

The Affine-Gap Global Alignment uses three dynamic programming matrices:

- $M(i,j)$: Best score ending with x_i matching y_j.
- $I_x(i,j)$: Best score ending with y_j aligned to a gap.
- $I_y(i,j)$: Best score ending with x_i aligned to a gap.

The recurrences are:

- $$I_x(i,j) = \max \begin{cases} M(i,j-1) - d \\ I_x(i,j-1) - e \end{cases}$$
- $$I_y(i,j) = \max \begin{cases} M(i-1,j) - d \\ I_y(i-1,j) - e \end{cases}$$

- $$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_i, y_j) \\ I_x(i-1,j-1) + s(x_i, y_j) \\ I_y(i-1,j-1) + s(x_i, y_j) \end{cases}$$

7.7 Profile Hidden Markov Models

A Profile HMM represents a multiple sequence alignment (MSA) probabilistically. It uses three classes of states for n alignment positions:

- **Match States** ($M_1 \dots M_n$): Correspond to conserved columns. Emits symbols based on a learned distribution.
- **Delete States** ($D_1 \dots D_n$): Mute states representing the deletion of a conserved position.
- **Insert States** ($I_0 \dots I_n$): Emit amino acids inserted between conserved positions.

For a model of length n, the architecture yields:

- Number of states: $n(\text{match}) + n(\text{deletion}) + (n+1)(\text{insertion}) + 2(\text{Begin/End}) = 3n + 3$.
- Number of transitions: $9n + 3$.

Probability smoothing via Laplace Correction (pseudocounts) is required. For an emission probability from state M_j:

$$e_{M_j}(a) = \frac{\text{count}(a) + 1}{\text{total valid symbols} + |\Sigma|} \tag{127}$$

Transition probabilities mapped from standard sequence alignment gaps are formulated as:

- $\log a_{MI} + \log a_{DI} \approx$ gap open penalty.
- $\log a_{II} \approx$ gap extension penalty.

Profile HMM Alignment via Dynamic Programming (Viterbi Adaptation): To score a new sequence $x_1 \dots x_L$ against the Profile HMM, log-odds scores are computed relative to a random model probability $\Pr(x_i)$. Let $\nu_j^M(i)$, $\nu_j^I(i)$, and $\nu_j^D(i)$ be the optimal path scores terminating at position i in state M_j, I_j, and D_j respectively:

- **Match Recurrence:**

$$\nu_j^M(i) = \log \frac{e_{M_j}(x_i)}{\Pr(x_i)} + \max \begin{cases} \nu_{j-1}^M(i-1) + \log a_{M_{j-1},M_j} \\ \nu_{j-1}^I(i-1) + \log a_{I_{j-1},M_j} \\ \nu_{j-1}^D(i-1) + \log a_{D_{j-1},M_j} \end{cases}$$

- **Insert Recurrence:**

$$\nu_j^I(i) = \log \frac{e_{I_j}(x_i)}{\Pr(x_i)} + \max \begin{cases} \nu_j^M(i-1) + \log a_{M_j,I_j} \\ \nu_j^I(i-1) + \log a_{I_j,I_j} \\ \nu_j^D(i-1) + \log a_{D_j,I_j} \end{cases}$$

- **Delete Recurrence (Mute state, no emission):**

$$\nu_j^D(i) = \max \begin{cases} \nu_{j-1}^M(i) + \log a_{M_{j-1},D_j} \\ \nu_{j-1}^D(i) + \log a_{D_{j-1},D_j} \end{cases}$$

Finally, because Negative Log Likelihood (NLL) scores heavily depend on sequence length, normalization to a Z-score is performed based on the empirical mean μ and standard deviation σ across sequences of matching length:

$$Z = \frac{|S - \mu|}{\sigma} \tag{128}$$

A score $Z > 4$ statistically confirms that the sequence is highly distinguishable from a background random sequence distribution.

8 Machine Learning in Bioinformatics: Neural Networks

8.1 Background and Foundations

Neural networks are utilized for both supervised and unsupervised learning. They are capable of handling both regression tasks, providing a real-value output, and classification tasks, providing a discrete output. The theoretical background of neural networks spans multiple disciplines:

- Neurology: Artificial intelligence seeks to utilize principles derived from biological neural systems.
- Statistics: It shares foundations with linear regression, generalized linear regression, and discriminant analysis.

8.2 The Neuron Model

A single neuron consists of input units, weights, a bias, a potential function, and an activation function. The input vector is denoted as $(x_1, \ldots, x_d)$. The weights associated with the inputs are $w_1, \ldots, w_d$, along with a bias unit fixed at 1 which is associated with weight w_0.

The potential a of the neuron is calculated as the weighted sum of its inputs

$$a = \sum_i w_i x_i \tag{129}$$

An activation function f is then used to convert the potential a into the final output o. The target output is denoted as y. For a simple linear neuron, the output is given by

$$o = f(a) \tag{130}$$

8.3 Activation Functions

Various activation functions can be applied to the potential a to introduce non-linearity or scale the output:

- Linear: $f(a) = a$
- Sigmoid: $f(a) = \frac{1}{1+e^{-a}}$
- Hyperbolic tangent (tanh): $f(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
- Rectified linear unit (ReLU):

$$f(a) = \begin{cases} 0 & \text{for } a < 0 \\ a & \text{for } a \geq 0 \end{cases}$$

8.4 Multi-Layer Neural Network Architecture

A Multi-Layer Perceptron (MLP) typically consists of input, hidden, and output layers. A two-layer neural network (containing one hidden layer and one output layer) utilizing a non-linear activation function acts as a universal function approximator. It can approximate any numeric function with arbitrary precision given an appropriate set of weights and sufficient hidden units. Historically, neural networks were limited to two or three layers (if the input is counted as a layer), with occasional benefits observed from increasing the layer count. This eventually expanded into deep learning architectures utilizing many layers.

The structure of a multi-layer neural network is defined as follows:

- **Input Layer**: Contains input units $x_0, x_1, \ldots, x_i, \ldots, x_d$, where $x_0 = 1$ is the bias. No activation function is applied at the input layer.
- **Hidden Layers**: Contains hidden units $z_0, z_1, \ldots, z_j, \ldots, z_M$, where $z_0 = 1$ is the bias. The inputs to this layer are connected via weights w_{ji}. The hidden layers use an activation function denoted as g, which can be linear, tanh, or sigmoid.
- **Output Layer**: Contains output units $y_1, \ldots, y_k, \ldots, y_c$ connected from the hidden layer via weights w_{kj}. The output layer uses an activation function f, which can be linear, sigmoid, or softmax.
- **Softmax Output**: For multi-class classification probabilities, the softmax function applied at the output layer is formulated as:

$$\text{Output}_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

8.5 Training Algorithm and Gradient Descent

The network is trained by adjusting weights using known examples from the training data. The objective is to minimize the difference between the network's actual output y and the desired target output t.

Dataset Definition: Given n training samples with target values $t^{(i)}$:

$$\{(x_1^{(1)}, x_2^{(1)}, \ldots, x_d^{(1)}, t^{(1)}), \ldots, (x_1^{(n)}, x_2^{(n)}, \ldots, x_d^{(n)}, t^{(n)})\} \tag{131}$$

Error Function: The goal is to minimize the sum of squared errors E with respect to the weights:

$$E = \sum_{i=1}^{n} (y^{(i)} - t^{(i)})^2 \tag{132}$$

Weight Update Rule: Weights are adjusted iteratively using gradient descent. If η denotes the learning rate and t denotes the time step (iteration), the update rule is:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}} \tag{133}$$

Backpropagation Algorithm Workflow:

- **Input**: Training data containing n samples with corresponding target values.
- **Initialization**: Initialize the network weights w.
- **Repeat** until a given number of iterations is reached or the error drops below a threshold:
  - For each data point x:
    - **Forward propagation**: Compute the outputs and activations throughout the network.
    - **Backward propagation**: Compute the errors for each output unit and hidden unit.
    - Compute the gradient of the error function with respect to each weight.
    - Update the weight using $w = w - \eta \frac{\partial E}{\partial w}$.

Update Modes:

- **Online training**: Weights are updated after evaluating each individual sample $x^{(i)}$.
- **Batch learning**: Weights are updated only after evaluating the entire training set.
- **Minibatch**: Weights are updated after evaluating a constant number B of training vectors.

8.6 Recurrent Neural Networks (RNN)

A Recurrent Network is fundamentally a series of feed-forward neural networks that share the exact same weights across time steps. They are particularly well-suited for processing time series data and sequence data, such as biological sequences and stock series.

The processing occurs sequentially:

- **Forward Pass**: At time 1, present input x_1; at time 2, present input x_2 along with the previous output y_1.
- **Backward Pass**: At time t, back-propagate the errors; at time $t-1$, back-propagate utilizing both the output errors of that step and the propagated errors from the previous step.

8.7 Overfitting and Regularization

Overfitting occurs when a network very closely models the training data, capturing accidental regularities and noise caused by sampling. Consequently, an overfitting function cannot generalize well to unseen data. A "good fitting" model captures the true regularities without matching every spurious data point.

To prevent overfitting, the network must have the correct capacity: enough to model true regularities, but not enough to model spurious regularities (assuming they are weaker). Standard techniques to limit a neural network's capacity include:

- Limiting the number of hidden units in the network.
- Limiting the absolute value of the weights, a technique known as weight decay.
- Stopping the learning process early before the model has time to overfit.

A fundamental practice is dividing the total dataset into three distinct subsets:

1. **Training data**: Used directly for learning the parameters of the model.
2. **Validation data**: Used for deciding the specific type of model and determining what amount of regularization works best.
3. **Test data**: Used to estimate how well the fully trained network works. It is expected that this estimate will be worse than the performance observed on the validation data.

8.8 Optimization: Speeding up Learning

Four distinct methods can be employed to accelerate the learning process:

1. **Use an adaptive global learning rate**: Slowly increase the rate if the learning process is not diverging, and quickly decrease the rate if it starts to diverge.
2. **Use a separate adaptive learning rate on each connection**: Adjust these individual rates based on the consistency of the gradient along that specific weight axis.
3. **Use momentum**: Instead of using the computed gradient to directly change the position of the weight "particle", use the gradient to change its velocity.
4. **Use a stochastic estimate of the gradient**: Calculate the gradient from a small subset of cases. This is highly effective on large, redundant datasets.

8.9 Problems of Neural Networks

Despite their capabilities, neural networks suffer from several inherent problems:

- Vanishing gradients during training.
- They cannot natively use unlabeled data.
- It is hard to understand or interpret the specific relationship between input and output.
- They cannot generate data.

8.10 Advanced Architectures: Deep AutoEncoders and Convolutional Neural Networks

Deep AutoEncoder Architecture: An autoencoder compresses high-dimensional data down to a bottleneck and then reconstructs it. A specific architecture described involves:

- Input [01]: 8281 units
- Sigmoid Layer: 50×50 dimension
- Sigmoid Layer: 25×25 dimension
- Sigmoid Layer: 10×10 dimension
- Bottleneck: 50 units
- Sigmoid Output: 8281 units

Deep Convolutional Neural Network (CNN): CNNs utilize shared convolutional and pooling layers to process spatial data. The specified architecture defines:

- **Shared CNN layers**: Sequential application of convolutional and max-pooling operations.
- **Hyperparameters**: Convolutional kernel stride is set to 1; max-pool kernel stride is set to 2.
- **Filter specifications and feature map shapes**: Includes transformations such as 9×9 convolutions, 3×3 pooling, 5×5 convolutions, yielding internal representations like 512×512, 32@104×104, 32@52×52, 32@24×24, 32@10×10, 3@112×112, $7 \times 8 \times 8$, and $8 \times 8 \times 8$.
- **Network Branches**: The shared representations branch into specialized heads for Part Detection, Joint Detection, and Joint Regression.

9 Machine Learning in Bioinformatics: Neural Network Applications

9.1 Proteins and Structural Hierarchy

- **Protein Definition**: A protein is defined as a chain of amino acids joined together by peptide bonds.
- **Amino Acid Structure**: An amino acid consists of a central alpha carbon (C_α), an amino group (NH_2), a carboxyl group ($COOH$), a hydrogen atom (H), and a variable side chain (R).
- **Polypeptide Backbone**: The sequence of $N - C_\alpha - C$ atoms makes up the backbone of the protein. Each amino acid has two rotational degrees of freedom, denoted by angles ϕ and ψ. The angle between $C = O$ and $N - H$ is approximately $180°$.
- **Primary Structure**: The chemical structure of the polypeptide chain, representing the linear sequence of amino acid residues linked by peptide bonds.
- **Secondary Structure**: The folding of the molecule that arises by linking the $C = O$ and NH groups of the backbone together through hydrogen bonds.
- **Tertiary Structure**: The 3D structure of the molecule consisting of secondary structures linked by looser segments of the polypeptide chain and stabilized primarily by sidechain interactions.
- **Quaternary Structure**: The aggregation of separate polypeptide chains into the fully functional protein.

9.2 Secondary Structure Classes and Ramachandran Plot

- **Ramachandran Plot**: A plot of observed pairs of ϕ and ψ angles in a collection of known protein structures. It describes acceptable angle pairs for individual amino acids and helps determine secondary structure types.
- α **Helix**: Helices arise when hydrogen bonds occur between the $C = O$ group of the amino acid at position i and the NH group of the amino acid at position $i + k$, typically where $k = 4$ or 5. The pairs near $\phi = -60°$ and $\psi = -60°$ correspond to right-handed helices. A standard α helix has 3.6 residues per turn and a rise per residue of 1.5 Å. The pitch calculation is shown below

$$\text{Pitch} = 3.6 \times 1.5 \text{ Å} = 5.4 \text{ Å}$$

- β **Sheet**: Formed from multiple side-by-side β-strands, composed of 2 to 15 strands with approximately 6 amino acids per strand. Ramachandran pairs near $(-90°, 120°)$ correspond to β strands. They can be configured in a parallel or an anti-parallel state, with the anti-parallel state being more stable. Side chains point alternately above and below the plane of the sheet.
- **Loops and Turns**: Loops usually contain hydrophilic residues, are found on the surfaces of proteins, and connect α helices and β sheets. Loops with fewer than 5 amino acids are classified as turns. In β turns, the peptide chain reverses direction, and the carbonyl carbon of one residue is hydrogen-bonded to the amide proton of a residue three positions away.
- **Coil**: A region of secondary structure that is not definitively a helix, a β sheet, or a recognizable turn.

9.3 Protein Structure Determination Techniques

- **X-ray Crystallography**: Capable of determining structures of any size with high accuracy ($1 - 3$ Å). The technique computes structure by creating a crystal, bombarding it with x-rays to create a diffraction pattern, extracting phases, constructing an electron density map, fitting, and refining an atomic model.
- **Nuclear Magnetic Resonance (NMR) Spectroscopy**: Used for small to medium-sized structures in solution, yielding moderate accuracy.

9.4 Data Processing and Encoding for Neural Networks

- **Dataset Creation Algorithm**:
  1. Download proteins from the Protein Data Bank (PDB).
  2. Select high-resolution structures (< 2.5 Å determination).
  3. Remove proteins with chain-breaks defined by $C_\alpha - C_\alpha$ distances > 4 Å.
  4. Remove redundancy by filtering out similar sequences using BLAST such that sequence similarity between test and training datasets is strictly $< 25\%$.
  5. Assign secondary structure labels using the DSSP program, which computes structures from 3D coordinates.
- **One-Hot Encoding**: An amino acid is represented by 20 inputs of 0s and 1s.
- **Label Encoding**: The secondary structure sequence is mapped to 3 inputs: Helix (100), Extended strand (010), and Loop/coil (001).
- **Context Windowing**: Inputs to the neural network use a sliding window of size w across the sequence. To account for N- and C-terminal boundaries, an extra 'spacer' bit is appended, resulting in $(w \times 21)$ parameters.

9.5 PHD Approach Algorithm

- **Evolutionary Profiles**: Leverages multiple sequence alignments obtained via Position Specific Iterated BLAST (PSI-BLAST) searches against the non-redundant database to compile profile tables.
- **First Level Neural Network (Local Alignment)**: Operates on 13 adjacent positions. The input features for each of the 13 residues include 20 profile values, 1 spacer flag, 2 insertion/deletion rates, and 1 conservation weight. Global statistics are also provided as inputs: 20 frequencies for amino acid composition, 4 bins for protein length, and 8 distance bins from the window to the protein ends. The output layer has 3 units denoting probabilities of helix, strand, or coil.
- **Second Level Neural Network (Smoothing)**: Removes biological anomalies from the raw output, such as helices of length 1. It takes as input the 3 probability outputs from the first level network along with the spacer and conservation weight for each of the 13 residues in the window.
- **Jury Decision**: An arithmetic average is computed over 4 networks derived from all combinations of the first level and second level networks trained on either balanced or unbalanced class distributions. The final prediction is the unit with the maximal arithmetic value.

9.6 PSI-PRED Approach

- **Scoring Matrices**: Uses a Position Specific Scoring Matrix (PSSM) instead of a standard probability matrix. The sequence weighting is performed implicitly by PSI-BLAST.
- **Sigmoid Transformation**: The raw PSSM values are transformed into the range $[0, 1]$ before being input to the network

$$f(x) = \frac{1}{1 + e^{-x}} \tag{134}$$

- **First Network Architecture**: Receives a 15-residue window mapped to 15×20 scaled inputs plus terminal spacers, generating 315 total inputs. Features 75 hidden units and 3 outputs.
- **Second Network Architecture**: Receives a 15-residue window spanning the 3 outputs of the first network, producing 60 inputs in total. Features 60 hidden units and outputs the final 3-state prediction.

9.7 SSpro Approach

- **Probability Matrices**: Uses a probability matrix alongside information theory approaches to weight the sequences.
- **Algorithm Type**: Evaluates variable length sequence inputs utilizing a 1-Dimensional Recurrent Neural Network (1D-RNN) or Bi-directional Input Output Hidden Markov Model (BRNN).
- **RNN Update Rules**

$$s_t = \tanh(\mathbf{w}_x x_t + \mathbf{w}_{rec} s_{t-1}) \tag{135}$$

$$y = \tanh(\mathbf{w}_h s_t) \tag{136}$$

- **Variables for RNN Equations**:
  - x_t is the input data array at sequence step t.
  - s_t is the hidden state vector at sequence step t.
  - s_{t-1} is the recurrent hidden state from the previous sequence step.
  - $\mathbf{w}_x$ denotes the feed-forward weight parameters from input to hidden state.
  - $\mathbf{w}_{rec}$ denotes the recurrent weight matrix governing transitions between hidden states.
  - $\mathbf{w}_h$ denotes the weight parameters mapping hidden states to output predictions.
  - y denotes the final output prediction array.
- **Bidirectional Context**: Features extraction processes sequentially combine calculations from a Forward Recurrent Neural Network (evaluating towards the C-terminus) and a Backward Recurrent Neural Network (evaluating towards the N-terminus) into a centralized Feed Forward Neural Network.

9.8 1D & 2D Auxiliary Predictions

- **Solvent Accessibility**: A 1D predictive task classifying structural residues as either Exposed or Buried using neural networks and multiple alignments.
- **Disordered Region Prediction**: Utilizes a 1D-RNN architecture to map sequential coordinates into ordered or disordered categorizations.
- **Contact Map Prediction**: A binary classification objective to determine if two residues (i, j) are in contact (spatially close). Contact is declared if distance $(i, j) < 8$ Å.
- **Contact Map Algorithms**: Models such as 2D-Recursive Neural Networks and Support Vector Machines map sequence spaces onto 2D distance threshold matrices.
- **Contact Map Input Encoding**: The classification algorithm processes a vector of approximately 400 features mapping pairs of residues to values between 0 and 1. Fundamental parameters include:
  - **Residue identity**: 20 binary numbers (e.g., one-hot encoding).
  - **Secondary structure**: 3 numbers (Helix mapped to 100, Strand to 010, Coil to 001).
  - **Solvent Accessibility**: 2 numbers (Exposed mapped to 10, Buried to 01).

9.9 AlphaFold Architectures

- **AlphaFold**: Uses neural network architectures that respect evolutionary, physical, and geometric constraints. Jointly embeds multiple sequence alignments (MSAs) alongside pairwise features. Implements an equivariant attention architecture, masked MSA loss, and utilizes intermediate losses to achieve iterative structural refinement.
- **AlphaFold 3**: Employs a diffusion-based architecture designed for the joint structure prediction of entire biomolecular complexes encompassing proteins, nucleic acids, small molecules, ions, and modified residues. Increases predictive accuracy for protein interactions with other molecule types by at least 50% relative to baseline methods.

10 Machine Learning in Bioinformatics: Graph Neural Networks

10.1 Graph Data and Representation

Graph data provides a superior representation for many complex data structures compared to traditional tabular formats, such as representing molecules (e.g., Cimetidine) via molecular graphs or heterogeneous networks like miRNA-disease networks. MicroRNAs (miRNAs) are defined as small, single-stranded, non-coding RNA molecules containing 21 to 23 nucleotides. They are intrinsically involved in RNA silencing and the post-transcriptional regulation of gene expression.

A graph is mathematically defined as $G = (V, E)$:

- V represents a set of nodes.
- E represents a set of edges.
- Tuples of the form (u, v) represent a directed edge.
- Pairs of the form $\{u, v\}$ represent an undirected edge.

Graphs are commonly represented using an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$. This matrix is strictly binary for an unweighted graph and contains continuous values for a weighted graph.

10.2 Assumptions for Graph Neural Networks

Graph Neural Networks (GNNs) operate under several core assumptions:

- Nodes are not independent and identically distributed (i.i.d.), as the framework inherently models an interconnected set of nodes.
- The architecture must find an encoding that explicitly preserves the underlying graph structure.
- The neighborhoods of nodes tend to exhibit and share similar attributes.

Furthermore, any neural network modeling a graph should preserve structural symmetries based on permutation matrices. Let $\mathbf{P} \in \{0, 1\}^{n\times n}$ be a permutation matrix satisfying $\mathbf{P1} = \mathbf{1}$ and $\mathbf{P}^T\mathbf{1} = \mathbf{1}$. The matrix operations possess the following properties:

- Applying $\mathbf{P}^T\mathbf{x}$ reorders the entries of a given vector $\mathbf{x}$.
- Applying $\mathbf{P}^T\mathbf{AP}$ consistently reorders the rows and columns of the matrix $\mathbf{A}$.

These properties lead to the derivations $\mathbf{P1} = \mathbf{1} \Rightarrow \mathbf{P}^T\mathbf{P1} = \mathbf{P}^T\mathbf{1} = \mathbf{1}$ and $\mathbf{P}^T\mathbf{1} = \mathbf{1} \Rightarrow \mathbf{PP}^T\mathbf{1} = \mathbf{P1} = \mathbf{1}$. Consequently, for a function f operating on an adjacency matrix $\mathbf{A}$ and an observation (feature) matrix $\mathbf{X}$ for each node, the network must be:

- **Permutation Invariant** (when computing a network prediction): $f(\mathbf{PAP}^T, \mathbf{PX}) = f(\mathbf{A}, \mathbf{X})$.
- **Permutation Equivariant** (when computing predictions for individual nodes): $f(\mathbf{PAP}^T, \mathbf{PX}) = \mathbf{P}f(\mathbf{A}, \mathbf{X})$.

10.3 Neural Message Passing

The primary goal of neural message passing is to combine the information derived from neighboring nodes to encode contextual graph information effectively. At every iteration, each node receives information from its neighbors, which is then systematically combined with its current features using a learnable function.

Given a graph $G = (V, E)$, let $\mathbf{X} \in \mathbb{R}^{|V| \times d}$ denote the initial node features, $\mathbf{h} = \{\mathbf{h}_1, \ldots, \mathbf{h}_N\}$ denote the hidden embeddings, and $\mathbf{z}_u$ denote the final node embedding for all $u \in V$. The superscript (k) is used to denote the computed value after iteration k. For a target node u, the iteration k of a GNN executes the following logic:

$$\mathbf{m}_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{\mathbf{h}_v^{(k-1)} \mid v \in \mathcal{N}(u)\right\}\right) \tag{137}$$

$$\mathbf{h}_u^{(k)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right) \tag{138}$$

After K iterations, the output corresponding to the final layer fundamentally defines the structural embedding of node u, given by $\mathbf{z}_u = \mathbf{h}_u^{(K)},\ u \in V$.

Iterative Message-Passing Pseudocode:

- **Input:** Undirected graph $G = (V, E)$.
- **Input:** Initial node embeddings $\{\mathbf{h}_u^{(0)} = \mathbf{x}_u\}$.
- **Input:** AGGREGATE$(\cdot)$ function.
- **Input:** UPDATE$(\cdot, \cdot)$ function.
- **Workflow:** For $l \in \{0, \ldots, L-1\}$ do:
- $\quad \mathbf{z}_u^{(l)} \leftarrow \text{AGGREGATE}\left(\left\{\mathbf{h}_v^{(l)} \mid v \in \mathcal{N}(u)\right\}\right)$.
- $\quad \mathbf{h}_u^{(l+1)} \leftarrow \text{UPDATE}\left(\mathbf{h}_u^{(l)}, \mathbf{z}_u^{(l)}\right)$.
- **Output:** Return final node embeddings $\left\{\mathbf{h}_u^{(L)}\right\}$.

10.4 The Basic Graph Neural Network

Message passing mathematically corresponds to a Multilayer Perceptron featuring linear combinations followed by a nonlinear transfer. The sequence consists of three distinct steps: summing the messages coming from all neighbors, combining the aggregated sum with the node's previous embedding, and applying a designated nonlinearity. This node-level operation is formalized as follows:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^k \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right) \tag{139}$$

The mathematical variables in the base GNN equation are rigidly defined as:

- $\mathbf{h}_u^{(k-1)} \in \mathbb{R}^{d^{(k-1)}}$: The vector denoting node embeddings from the previous layer.
- $\mathbf{W}_{\text{self}}^k, \mathbf{W}_{\text{neigh}}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$: Learnable weight matrices operating on the node's current state and its neighborhood aggregate, respectively.
- $\mathbf{b}^{(k)} \in \mathbb{R}^{d^{(k)}}$: The learnable bias term vector.

- $\sigma : \mathbb{R} \to \mathbb{R}$: A standard nonlinear transfer function (e.g., tanh, sigmoid, or ReLU) applied element-wise.

These parameter sets (weights and biases) can either be shared iteratively across GNN message passing iterations or selectively trained separately for each designated layer. While outlined as node-level operations, identical mathematical principles can similarly be applied to process edges.

10.5 Taxonomy and Methodology of GNNs

The overarching architecture of Graph Neural Networks branches out into distinct methodological families:

- **Convolutional Networks:** Encompasses both Spectral Methods (e.g., Graph Conv Network (GCN), ChebNet) and Spatial Methods (e.g., Graph Attention Network (GAN), Learnable GCN).
- **Recurrent Networks:** Features Convergence-based methods (e.g., Lagrangian Propagation GNN, Graph Echo State Network) and Gate-based architectures (e.g., Gated GNN (GGNN), Graph LSTM).
- **Skip Connections:** Includes architectures like Jump Knowledge Network (JKN) and Highway GCN.

Specifically for Convolutional GNNs, the methodological approaches diverge into two paradigms:

- **Spatial approach:** This methodology defines convolutions directly on the graph strictly based on the underlying graph topology.
- **Spectral methods:** An input graph signal $\mathbf{X}$ is transformed directly into a spectral domain by the graph Fourier transform $\mathcal{F}$, after which the convolution operation is explicitly performed. Post-convolution, the signal is deterministically transformed back into the spatial domain utilizing the inverse Fourier transform $\mathcal{F}^{-1}$. Spectral methods natively require at most $\mathcal{O}(n)$ parameters per generated feature map, which allows for highly efficient forward propagation scaling.

10.6 Downstream Machine Learning Tasks

GNNs are leveraged extensively across several distinct structural tasks within Bioinformatics and broader machine learning scopes:

- **Link Prediction:** Intended to forecast the presence of links definitively between two entities. In the context of social networks, it remains crucial for accurately inferring targeted social interactions or systematically recommending potential friends to participating users.
- **Graph Clustering:** Involves grouping sets of nodes forming closely interconnected regions, effectively utilizing either strict edge weights or computed distances. This can also extend to grouping entire discrete graphs as objects to be clustered, systematically grouping them strictly based on their quantifiable similarity metrics.
- **Node Classification:** Aims to reliably assign labels to target nodes strictly by examining the labels of their encompassing "neighbors". Typically, these explicit problems are heavily trained using a semi-supervised approach, meaning that strictly only a localized portion of the overarching graph is labeled initially.

11 Introduction to Explainable AI

11.1 The Need for Explainability

In machine learning, there is a fundamental distinction between two types of systems:

- **Black box AI:** Systems where there is no understanding of the internal decision-making process.
- **Explainable AI (XAI):** Systems designed to allow users to understand the model's inner workings.

11.2 Explainability vs. Accuracy Trade-off

There is a general trade-off between a model's explainability and its predictive performance:

- **Basic models** (e.g., Rule-based systems, Linear models, Decision Trees) have lower model complexity. They are more inherently explainable but generally less precise.
- **Complex models** (e.g., K-Nearest Neighbors, Support Vector Machines, Neural Networks) have higher model complexity. They are more precise but less explainable.

For instance, Decision Trees explicitly show a decision path based on distinct conditions, making them transparent. In contrast, Neural Networks derive patterns through interconnected layers (student features mapped to accepted/rejected outcomes) and are not inherently transparent.

11.3 Categories of Explainability Techniques

Explainability techniques can be categorized into:

- Model-specific vs. model-agnostic techniques.
- Local (explaining individual predictions) vs. global (explaining the entire model) methods.

Advanced topics in this domain include explainability metrics, unsupervised explainability, and generative AI explainability.

11.4 Inherently Explainable Models

11.4.1 Linear Models

Linear models are inherently explainable because they learn a linear combination of input features. The relationship is defined as:

$$\text{Output} = c_0 + c_1 \cdot \text{feature}_1 + c_2 \cdot \text{feature}_2 + \cdots + c_n \cdot \text{feature}_n \tag{140}$$

The coefficients (c_i) indicate the importance of each corresponding feature:

- A higher absolute value indicates higher importance.
- A lower absolute value indicates lower importance.
- **Note:** Feature scales must be normalized before computing and comparing coefficients.

11.4.2 Tree-Based Models

- **Decision Tree:** Used for both regression and classification. It utilizes a tree-like structure for predictions where each decision split is based on a single feature. This structure makes it inherently explainable.
- **Random Forest:** An ensemble of many decision trees, used for regression and classification. The aggregation of multiple trees complicates direct interpretability. Feature importance in a random forest is measured by the reduction of uncertainty in predictions, which is fundamentally different from the weight coefficients in linear models.

11.5 Model-Agnostic Techniques

11.5.1 Permutation Importance

Permutation importance is a model-agnostic method used to assess feature importance by measuring the effect of feature shuffling on a model's performance. It is highly versatile.

Workflow of Permutation Importance:

1. Evaluate the trained machine learning model on a dataset to determine the baseline performance, denoted as P_1.
2. Shuffle the values of a single target feature (e.g., x_3) across the dataset, destroying its relationship with the target variable.
3. Re-evaluate the trained model on this modified dataset to determine the shuffled performance, denoted as P_2.
4. Calculate the importance of the shuffled feature as the difference in performance:

$$\text{Importance} \approx P_1 - P_2$$

5. **Interpretation:** A small drop in performance implies the feature is insignificant. A large drop in performance indicates the feature is significant.

11.5.2 LIME (Local Interpretable Model-Agnostic Explanations)

LIME explains the predictions of complex models by working on individual instances. It is agnostic to the underlying model type.

LIME builds an interpretable surrogate model defined as:

$$g(z) = w_0 + \sum_{i=1}^{d} w_i z_i \tag{141}$$

Where:

- g is a simple, interpretable model (usually linear).
- z_i are the interpretable features.
- w_i are the weights that indicate feature importance.

This local model approximates the original complex model only in the immediate vicinity of the target data point.

Workflow of LIME:

1. Choose a specific prediction to explain (one sample x).

2. Generate random perturbations around the sample: $x \Rightarrow x'_1, \ldots, x'_n$.

3. Query the original black-box model to compute the predictions for the perturbed samples: $f(x'_i)$.

4. Weight the generated samples by their proximity to the original sample x. Nearby samples receive larger weights. Usually, an exponential kernel is used as the distance metric.

5. Fit the interpretable model by training a simple weighted regression (such as a linear model, a sparse linear model, or a small decision tree).

6. The resulting coefficients of the simple model become the local explanation.

Objective Function: LIME optimizes the following objective function to find the best explanation:

$$\text{Explanation} = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{142}$$

Where:

- f is the original black-box model.
- g is the interpretable surrogate model.
- L is the local approximation error (how unfaithful g is to f in the locality defined by π_x).
- π_x is the locality weighting (defining the neighborhood around x).
- $\Omega(g)$ is the complexity penalty (ensuring the surrogate model g remains simple and interpretable).

LIME Explainers by Data Type: LIME utilizes specific explainers tailored to tabular, text, and image data.

- **LIME for Text:** Finds how each word impacts the prediction. It perturbs the input by removing words and observes the resulting probability changes (e.g., the word "excellent" strongly increases positive sentiment, while "terrible" strongly decreases it).
- **LIME for Images:** Divides the image into superpixels, randomly masks these regions, and observes which regions affect the classification the most. It then highlights the regions that are most important for the prediction.

Important Limitations of LIME:

- **Local Scope:** Different nearby regions may produce different explanations. LIME explains only the behavior near one specific datapoint; it does NOT explain the global logic of the model.
- **Instability:** Because perturbation sampling is random and local regression can be unstable, two separate runs of LIME on the same datapoint can produce slightly different explanations.
- **Neighborhood Sensitivity:** The explanation depends heavily on how perturbations are generated, the distance metric used, and the kernel width. A poor neighborhood design leads to misleading explanations.

11.5.3 SHAP (SHapley Additive exPlanations)

SHAP is a model-agnostic technique utilizing Shapley values from cooperative game theory to explain the output of any machine learning model. It connects optimal credit allocation with local explanations, quantifying individual feature contributions to a specific prediction.

Cooperative Game Theory Perspective:

- **Original Framework:** Players cooperate to form a coalition, which then receives a reward.
- **Machine Learning Reinterpretation:** Players represent the **features**, the coalition represents a **subset of features**, and the reward is the **model prediction**.

The core challenge SHAP addresses is that features interact nonlinearly and their contributions depend entirely on context (which other features are present).

Shapley Value Formula: For a feature subset S, the coalition value function $v(S)$ is defined as the expected model output using only the features in S:

$$v(S) = \mathbb{E}[f(X) \mid X_S] \tag{143}$$

When feature i joins the coalition S, the change in prediction (the marginal contribution) is calculated as:

$$v(S \cup \{i\}) - v(S) \tag{144}$$

Because this contribution depends on the features already present in S, the Shapley value averages these marginal contributions over all possible coalitions and feature orderings:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[v(S \cup \{i\}) - v(S) \right] \tag{145}$$

Where ϕ_i is the Shapley value (average marginal contribution) for feature i, weighted fairly over all permutations of features.

Fairness Axioms of SHAP: Shapley values are the only attribution method guaranteed to satisfy the following game-theoretic axioms:

1. **Efficiency:** All feature contributions sum exactly to the difference between the actual prediction and the expected baseline prediction (complete attribution, nothing is left unexplained).
$$\sum_i \phi_i = f(x) - \mathbb{E}[f(X)]$$
2. **Symmetry:** If two features contribute identically to all possible coalitions, they receive equal attribution.
3. **Dummy feature:** If a feature changes nothing (its marginal contribution is always zero), its Shapley value is zero.
4. **Additivity (or linearity):** Explanations remain linear and compositional. The explanation of a sum of models equals the sum of their individual explanations.

Kernel SHAP and Computational Complexity: Exact computation of Shapley values requires evaluating all possible subsets, which scales at $O(2^d)$. For a large number of features d, exact computation is mathematically infeasible. **Kernel SHAP** is used as an approximation.

Workflow of Kernel SHAP:

1. Sample various feature subsets.

2. Evaluate the original model on these masked versions of the input.

3. Fit a weighted linear regression that approximates the model locally:

$$g(z') = \phi_0 + \sum_{i=1}^{d} \phi_i z'_i$$

   Where $z'_i \in \{0, 1\}$ (1 means the feature is present, 0 means the feature is absent).

4. Use the resulting regression coefficients as the approximated Shapley values.

SHAP vs. LIME Comparison: Kernel SHAP resembles LIME but differs fundamentally. Kernel SHAP can be viewed as "LIME with a theoretically justified kernel."

SHAP	**LIME**
Game-theoretic attribution	Heuristic local approximation
Shapley-derived weighting	Flexible weighting
Satisfies Shapley axioms	No fairness guarantees
More theoretically grounded	Usually faster

11.6 Explainability Metrics

To evaluate the quality of an explanation, several metrics analyze if the explanation is correct, stable, reflects real model behavior, and is understandable to humans.

- **Fidelity (Faithfulness):** Measures how accurately the explanation matches the actual model. It evaluates if the features identified as important truly influence the model's predictions. Low faithfulness misleads user trust. It is calculated by taking the absolute difference between the model's prediction on the original sample and the modified sample (altered based on the explanation):

$$\text{Faithfulness} = |\text{New prediction} - \text{Original prediction}| \tag{146}$$

- **Stability / Robustness:** Measures the explanation's sensitivity to small, non-meaningful perturbations in the input.

- **Sparsity / Simplicity:** Measures human interpretability (fewer, clearer features are better).

- **Consistency:** Assesses the stability of explanations when a model is trained on different subsets of the same dataset. Low consistency means no robust explanations can be formed. It is calculated using the Cosine similarity between the feature importance vectors of the two subsets, mapped on a scale of $[-1, 1]$:
  - -1: Opposite explanations.
  - 0: No consistent explanations.
  - $+1$: Highly consistent explanations.

- **Completeness:** Evaluates whether the explanation fully accounts for the prediction.

- **Human-centered metrics:** Measures the practical usefulness of the explanation to end users.

12 Machine Learning in Bioinformatics: Protein Structure and AlphaFold

12.1 Protein Structure and the Folding Problem

Proteins are biological macromolecules composed of linear chains of amino acids that fold into specific three-dimensional structures dictating their function. The structural hierarchy is defined in four levels:

- Primary: The linear sequence of amino acids.
- Secondary: Local structural elements such as α-helices and β-sheets.
- Tertiary: The overall three-dimensional conformation.
- Quaternary: Complexes formed by multiple interacting protein subunits.

12.1.1 Levinthal's Paradox and Energy Landscapes

The central challenge of structural biology is determining how a linear sequence specifies a unique 3D structure. Levinthal's paradox highlights the exponential complexity of this search. If folding were a random search where each bond adopts only a few possible angles, the total number of conformations would scale exponentially. For a modest protein, the search space can exceed:

$$\mathcal{O}(3^{100}) \tag{147}$$

Even if conformations were sampled at 10^{-13} seconds per configuration, exhaustive search would take longer than the age of the universe. The resolution to this paradox relies on an energy landscape, often modeled as a funnel:

- High-energy states correspond to unfolded structures.
- Proteins fold following highly directed kinetic pathways driven by a structured landscape with a gradual bias toward low-energy native states.
- Folding is hierarchical: local structures form quickly, constraining the global search space, before long-range interactions refine the final state.
- The process is physically guided by hydrophobic interactions, hydrogen bonding, electrostatics, and entropy-enthalpy trade-offs.

12.2 Classical Methods for Structure Determination

Prior to modern machine learning methods, protein structures were resolved experimentally via three primary techniques:

- **X-ray Crystallography:** Requires crystallized proteins to produce electron density maps. It offers typical resolution ranging from $\sim$ 1–3 Å.
- **Nuclear Magnetic Resonance (NMR) Spectroscopy:** Analyzes proteins in solution by measuring the resonance of atomic nuclei under a magnetic field. The absorption frequency shifted by surrounding electrons yields a chemical shift, while spin-spin coupling and the Nuclear Overhauser Effect (NOE) provide distance constraints. It is primarily limited to smaller proteins under $\sim$ 30–40 kDa.

- **Cryo-Electron Microscopy (Cryo-EM):** Captures random orientations of rapidly frozen samples in vitreous ice, reconstructing 3D structures computationally via particle picking and averaging. It achieves $\sim$ 2–4 Å resolution and is highly effective for large macromolecular complexes.

The Critical Assessment of protein Structure Prediction (CASP) serves as a benchmark competition evaluating structural predictions using metrics such as the Global Distance Test (GDT) and Root Mean Square Deviation (RMSD).

12.3 AlphaFold 1 (First Generation)

Introduced in 2018, the first generation of AlphaFold utilized deep residual convolutional neural networks predicting geometric constraints rather than raw spatial coordinates.

12.3.1 Distogram Prediction and Optimization

The network operates on features extracted from Multiple Sequence Alignments (MSA) and co-evolution signals, where residues mutating together imply spatial proximity.

- The model outputs a distance map, or "distogram".
- For each residue pair (i, j), it predicts a probability distribution over discrete distance bins.
- The predicted distance distributions are subsequently converted into a continuous potential energy function.
- The 3D coordinates are derived by optimizing this energy function using gradient-based descent.
- The network applies a cross-entropy loss function over the distance bins through supervised learning on known structures from the Protein Data Bank (PDB).

12.4 AlphaFold 2 (End-to-End Prediction)

AlphaFold 2 reinvented the pipeline into a fully end-to-end deep learning system directly predicting 3D atomic coordinates. The architecture is divided into two primary deep learning blocks: the Evoformer and the Structure Module.

12.4.1 The Evoformer Architecture

The Evoformer replaces independent sequence tracking with iterative representations that mutually update one another over 48 network blocks. It maintains two primary numerical representations:

- An **MSA Representation** containing evolutionary data encoded as a matrix of dimensions $(N_{\text{seq}} \times N_{\text{res}} \times d)$.
- A **Pair Representation** capturing $N_{\text{res}} \times N_{\text{res}}$ relationships (interactions between residues i and j), dimensionality $(N_{\text{res}} \times N_{\text{res}} \times d)$.

To compute interactions, the MSA representation utilizes attention mechanisms factorized into row-wise and column-wise operations.

- Row-wise attention processes residues within sequences, augmented by a spatial bias extracted directly from the Pair Representation to identify interacting amino acids.
- Column-wise attention compares the same position across different aligned sequences.

12.4.2 Triangular Geometric Updates

Because pairwise distances must obey the metric properties of 3D space, the Evoformer applies Triangle Operations to enforce consistency upon residue triplets (i, j, k). If the distance boundaries $d(i, j)$ and $d(j, k)$ are constrained, then $d(i, k)$ must mathematically agree.

- **Triangle Multiplication:** The state of $\text{Pair}(i, k)$ is updated via a message-passing sum over intermediate nodes j.
- The multiplication accumulates contributions from all valid paths: $\text{Pair}(i, j) \times \text{Pair}(j, k) \rightarrow \text{Pair}(i, k)$.
- **Triangle Attention:** Updates the edge $\text{Pair}(i, j)$ by calculating attention weights over third-party nodes k, summing their weighted contributions to deduce the (i, j) spatial relation.

12.4.3 The Structure Module

The Structure Module interprets the final representations, treating the amino acid backbone as a "residue gas".

- Each amino acid backbone segment (Nitrogen, C_α, and Carbon) is abstracted as a rigid triangle floating independently in the origin.
- Transformations mapping these rigid bodies into 3D space are parameterized as 4×4 affine matrices.

The affine transformation matrix $\mathbf{M}$ is composed of a 3×3 rotation matrix (a_{ij}) and a 1×3 displacement column $(t_x, t_y, t_z)^\top$ to execute rigid-body motion:

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{148}$$

- The module utilizes **Invariant Point Attention (IPA)**, which guarantees the network's predictions remain invariant to external global rotations and translations.
- IPA relies on calculating the L2 norm between local points, establishing an intrinsic frame for structural data.
- Side-chain conformations are represented via scalar torsion (dihedral) angles ϕ, ψ, χ, which are explicitly translated into final 3D coordinates via geometric routines.

12.4.4 Loss Functions and Confidence Metrics

AlphaFold 2 implements a series of auxiliary objectives alongside the primary structural loss:

- **Frame Aligned Point Error (FAPE):** Evaluates predicted atomic coordinates against true positions; crucially, FAPE penalizes reflection symmetries to strictly enforce the correct molecular chirality of the protein chain.
- **Distogram Loss:** Constrains distance distributions between residues.
- The system outputs two major confidence estimations: the predicted local distance difference test (pLDDT) and the predicted aligned error (PAE).

12.5 AlphaFold 3 (Multimodal Biomolecular Prediction)

AlphaFold 3 expands capabilities to predict complex assemblies including protein-protein interactions, protein-DNA/RNA binding, and ligand structures.

12.5.1 Architecture Simplification and Pairformer

To improve computational scalability, the computationally intensive Evoformer is replaced with a streamlined Pairformer module. This substitution massively limits explicit MSA processing, relying more heavily on singular representations.

12.5.2 Generative Diffusion Module

AlphaFold 3 abandons the rigid-body residue-gas operations (IPA and side-chain angle generation) in favor of a generative diffusion network directly predicting raw atomic coordinates.

- **Forward Diffusion:** A deterministic Markov process incrementally adds Gaussian noise to the true atomic coordinates over sequential time steps t, eventually deteriorating the structure into pure noise.
- **Reverse Denoising Process:** The model is trained to reverse this corruption by learning the target's data probability distribution.
- At step t, the network takes the noisy input vector x_t and attempts to predict the clean coordinate tensor x_{t-1}:

$$x_{t-1} = x_t - \text{predicted noise} \tag{149}$$

- Denoising at large scales resolves global topology, while denoising at smaller noise margins resolves precise local stereochemistry.
- This continuous formulation accommodates arbitrary chemical components without explicitly relying on domain-specific bonding logic or stereochemical penalty losses.