Lecture 8:

ML Fundamentals for Biomedical Data

ML meets medicine - Success stories & Challenges

Introduction to Biomedical Data Science

Lecture Contents

Part 1/3:

Biomedical Data Challenges

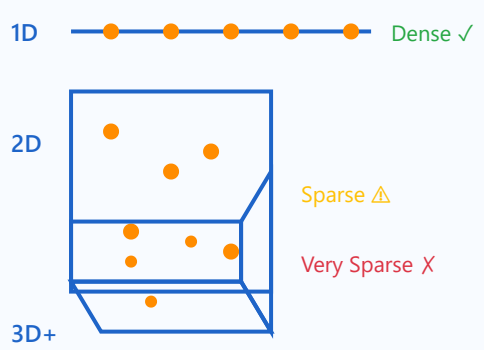
- Unique characteristics of biomedical data
- Statistical considerations
- Preprocessing strategies

High Dimensionality

⚠ The Curse of Dimensionality

When features (P) greatly exceed samples (N): P >> N

Data Sparsity in High-D



Overfitting Risks

Models memorize training data rather than learning generalizable patterns

Distance Metrics Fail

All points appear equidistant in very high dimensions

Sparse Data Space

Data points become increasingly sparse in high-dimensional space

Computational Cost

Training time and memory requirements grow exponentially

✓ Solutions & Strategies

- Regularization (L1/L2)
- Feature Selection
- Dimensionality Reduction
- Cross-validation
- Domain Knowledge

Small Sample Sizes

📊 Statistical Power Challenge

Limited samples reduce ability to detect true effects and increase risk of false discoveries

Cross-validation

K-fold, stratified, leave-one-out strategies for robust evaluation

Bootstrap Methods

Resampling techniques to estimate uncertainty and confidence intervals

Data Augmentation

Generate synthetic samples while preserving statistical properties

Transfer Learning

Leverage pre-trained models from larger datasets or related domains

Regularization

Penalize model complexity to prevent overfitting on small datasets

Domain Priors

Incorporate biological knowledge to guide model learning

Class Imbalance

⚠ The Accuracy Paradox

99% accuracy is useless if 99% of samples are negative!
Model predicting all negatives achieves high accuracy but zero clinical utility

Sampling Strategies

- Random oversampling
- Random undersampling
- SMOTE (Synthetic Minority)
- ADASYN (Adaptive Synthetic)

Cost-sensitive Learning

- Weighted loss functions
- Class weights in sklearn
- Focal loss for deep learning
- Penalize misclassifications differently

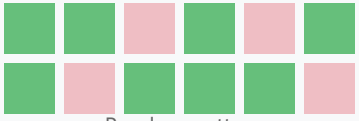
✓ Proper Evaluation Metrics

Precision	Recall	F1-score	PR-AUC
Balanced Accuracy	Matthews CC	Cohen's Kappa	G-mean

Missing Data

MCAR

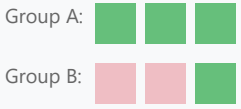
Missing Completely At Random - missingness unrelated to data



Random pattern

MAR

Missing At Random - related to observed variables



Depends on group

MNAR

Missing Not At Random - related to unobserved values

Missing if high value

Imputation Strategies

Mean/Median *Simple but biased*	**K-NN** *Uses similar samples*	**MICE** *Multiple Imputation*
MissForest *Random Forest based*	**EM Algorithm** *Maximum Likelihood*	**Deep Learning** *Autoencoders, GANs*

Batch Effects

⚠ Technical Variation Problem

Non-biological variations from different labs, instruments, or time periods can overwhelm true biological signals

Correction Methods

- ComBat (most popular)
- Limma removeBatchEffect
- Harmony (single-cell)
- Seurat CCA integration
- Deep learning approaches

Best Practices

- Randomize across batches
- Include batch in study design
- Balance classes per batch
- Use supervised correction
- Validate on independent data

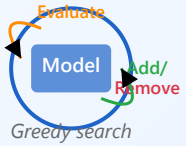
Feature Selection

Filter Methods

Independent of model

- Correlation analysis
- Chi-square test
- ANOVA F-test
- Mutual information

Wrapper Methods



Greedy search

- Forward selection
- Backward elimination
- Recursive Feature Elimination
- Genetic algorithms

Embedded Methods

Built into model

- Lasso (L1)
- Ridge (L2)
- Elastic Net
- Tree importance

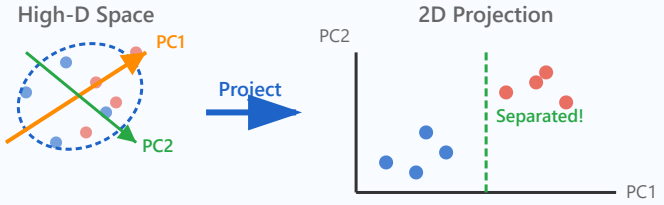
✓ Advanced & Clinical Considerations

Stability Selection | Permutation Importance | Clinical Interpretability | Domain Knowledge Integration

Dimensionality Reduction

PCA

Principal Component Analysis - linear transformation preserving maximum variance

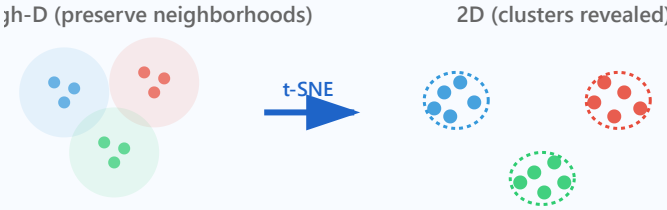


Clinical Applications:

→ Gene expression clustering
→ Quality control visualization
→ Batch effect detection

t-SNE

t-Distributed Stochastic Neighbor Embedding - nonlinear visualization method



Clinical Applications:

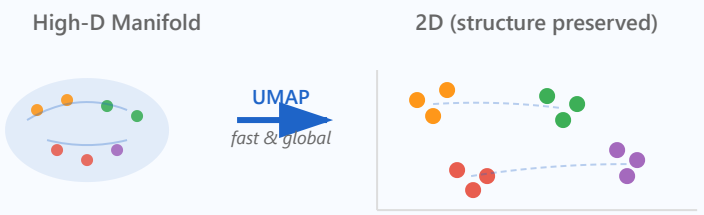
→ Single-cell RNA-seq visualization
→ Patient subtype discovery
→ Exploratory data analysis

UMAP

Uniform Manifold Approximation - faster than t-SNE, preserves global structure

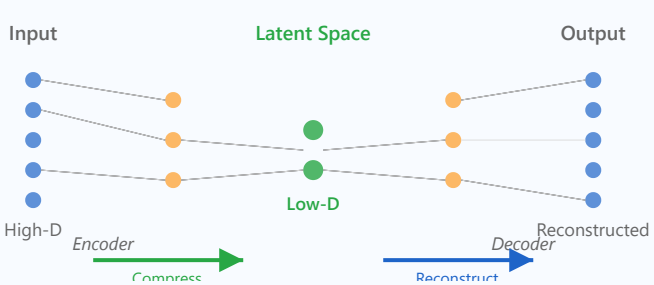
Autoencoders

Deep learning compression - learns nonlinear representations



Clinical Applications:

→ Large dataset visualization
→ Multimodal data integration
→ Trajectory inference



Clinical Applications:

→ Feature extraction for prediction
→ Denoising medical images
→ Anomaly detection

Part 2/3:

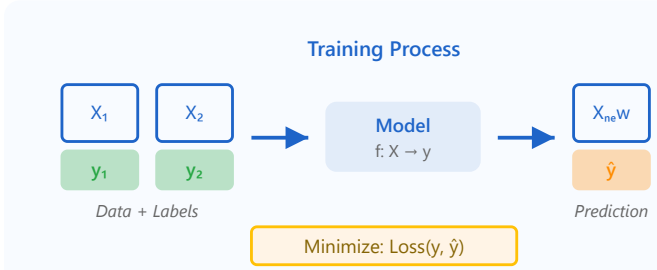
ML Methods for Biomedical Applications

- Algorithm taxonomy
- Model selection strategies
- Performance evaluation
- Interpretability requirements

Supervised vs Unsupervised Learning

Supervised Learning

Learn from labeled data - input-output pairs with known targets

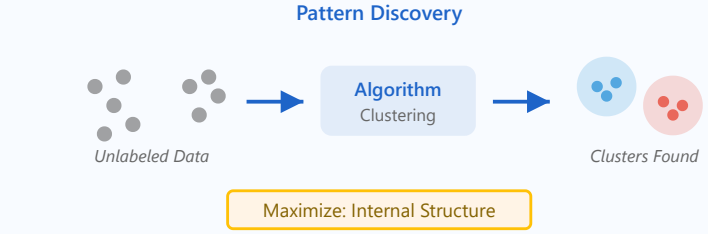


Clinical Examples:

- Disease diagnosis (healthy/sick)
- Drug response prediction
- Survival time estimation
- Risk score calculation

Unsupervised Learning

Discover patterns in unlabeled data - no predefined targets



Clinical Examples:

- Patient subtype discovery
- Gene expression clustering
- Anomaly detection
- Feature extraction

Hybrid Approaches

Semi-supervised Learning | Self-supervised Learning | Weakly-supervised Learning

Classification in Medicine

Common algorithms for diagnosis and prediction tasks

Logistic Regression

Linear model for binary/multi-class classification with probability outputs

- ✓ Highly interpretable
- ✓ Fast training
- ✗ Assumes linearity
- ✗ Limited complexity

Random Forests

Ensemble of decision trees for robust, non-linear classification

- ✓ Handles non-linearity
- ✓ Feature importance
- ✗ Less interpretable
- ✗ Can overfit

Support Vector Machines

Maximum margin classifier with kernel tricks for non-linear boundaries

- ✓ Effective in high-dim
- ✓ Versatile kernels
- ✗ Slow on large data
- ✗ Hard to interpret

Neural Networks

Deep learning models for complex pattern recognition

- ✓ Highest performance
- ✓ Automatic features
- ✗ Black box
- ✗ Needs large data

1. Logistic Regression

Overview: Logistic regression is a fundamental statistical method that models the probability of a binary outcome using a logistic (sigmoid) function. Despite its name, it's a classification algorithm that outputs probabilities between 0 and 1.

How It Works:

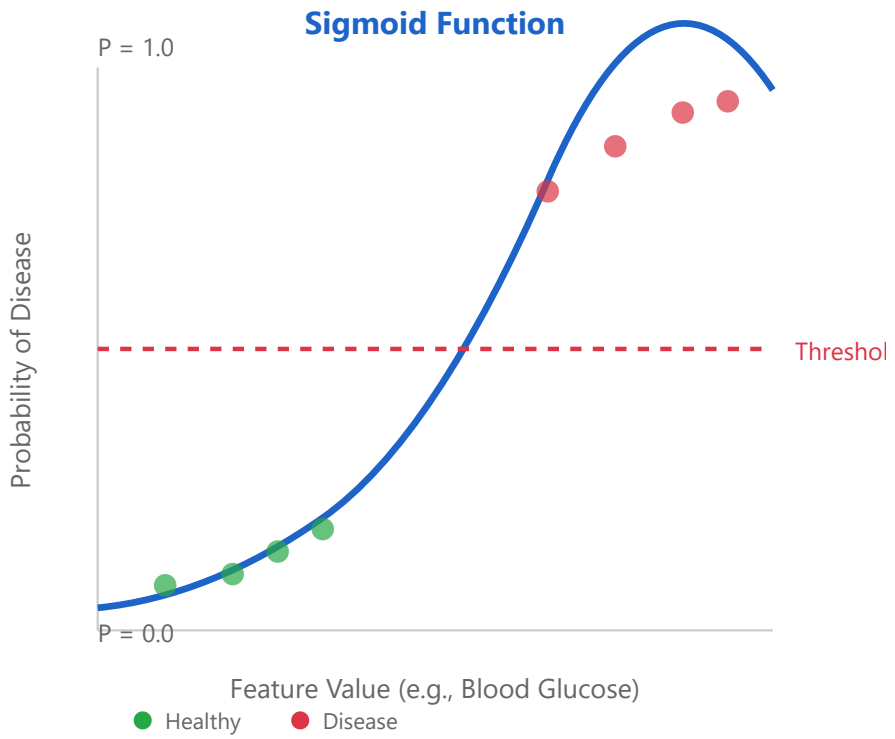
- Uses a linear combination of input features
- Applies sigmoid function: P(y=1) = 1 / (1 + e^(-z))
- Classifies based on probability threshold (typically 0.5)
- Optimized using maximum likelihood estimation

Medical Application Example:

Disease Risk Prediction: Predicting diabetes risk based on age, BMI, blood glucose, and family history. Each coefficient shows how much each factor increases or decreases the log-odds of diabetes. *Example:* If age coefficient = 0.05, each year increases diabetes odds by e^0.05 ≈ 1.05 times.

Key Strengths in Medicine:

- Provides interpretable odds ratios for clinical decision-making
- Fast to train and deploy in clinical systems
- Well-established statistical framework with confidence intervals
- Works well with limited data

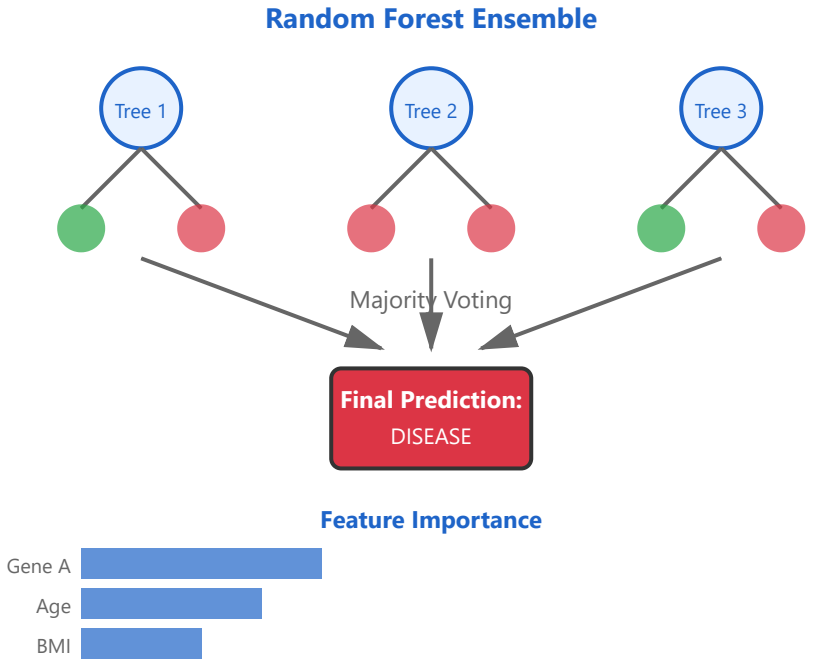


2. Random Forests

Overview: Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the classes from individual trees. It combines the predictions of many trees to reduce overfitting and improve accuracy.

How It Works:

- Creates multiple decision trees using bootstrap sampling
- Each tree uses a random subset of features at each split
- Aggregates predictions through majority voting (classification)
- Provides feature importance rankings



Medical Application Example:

Cancer Diagnosis: Classifying tumor types based on gene expression data, imaging features, and patient demographics. Random forests can handle hundreds of genes simultaneously and identify which genes are most important for classification. *Example:* Using 500 trees to analyze 1,000 genomic features, identifying the top 20 genes that best distinguish between benign and malignant tumors.

Key Strengths in Medicine:

- Handles complex, non-linear relationships in clinical data
- Robust to outliers and missing values
- Provides feature importance for biomarker discovery
- Minimal hyperparameter tuning required

3. Support Vector Machines (SVM)

Overview: Support Vector Machine is a powerful classification algorithm that finds the optimal hyperplane (decision boundary) that maximizes the margin between different classes. It can handle both linear and non-linear classification through kernel functions.

How It Works:

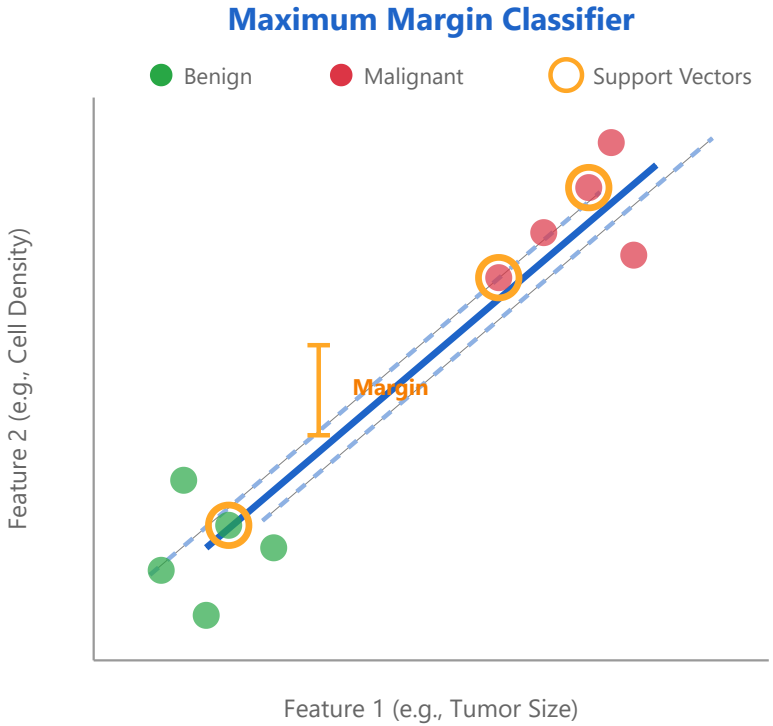
- Finds the hyperplane with maximum margin between classes
- Uses support vectors (critical points closest to decision boundary)
- Applies kernel trick for non-linear transformations
- Optimizes using quadratic programming

Medical Application Example:

Medical Image Classification: Distinguishing between different types of brain lesions in MRI scans using texture features, intensity patterns, and spatial characteristics. SVMs excel with high-dimensional imaging data.
Example: Using an RBF kernel to classify brain tumors into gliomas, meningiomas, or metastases based on 200+ imaging features extracted from MRI scans.

Key Strengths in Medicine:

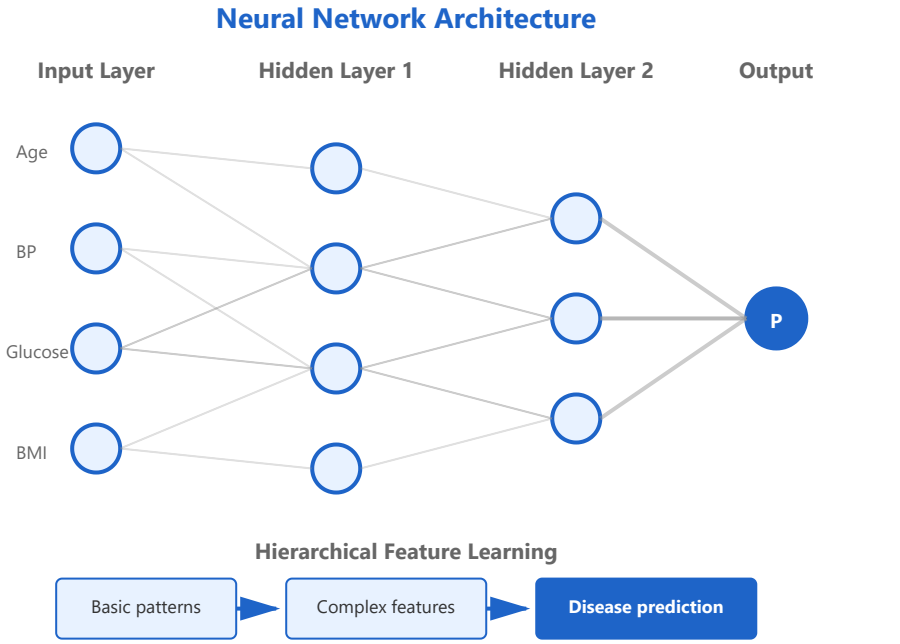
- Effective with high-dimensional medical data (genomics, imaging)
- Memory efficient (only uses support vectors)
- Flexible with different kernel functions for complex patterns



- Good performance with limited training samples

4. Neural Networks (Deep Learning)

Overview: Neural networks are computational models inspired by biological neural systems, consisting of interconnected layers of nodes (neurons) that can learn complex patterns through backpropagation. Deep learning uses multiple hidden layers to automatically extract hierarchical features.



How It Works:

- Input layer receives raw data (images, sequences, tabular data)
- Hidden layers extract progressively abstract features
- Weights adjusted through backpropagation and gradient descent
- Activation functions introduce non-linearity (ReLU, sigmoid, tanh)

Medical Application Example:

Chest X-ray Diagnosis: Convolutional Neural Networks (CNNs) analyze chest X-rays to detect pneumonia, tuberculosis, lung cancer, and other conditions. The network automatically learns to identify relevant patterns like consolidations, nodules, and infiltrates.

Example: A CNN with 50 layers trained on 100,000 chest X-rays achieving 95% accuracy in detecting pneumonia, comparable to expert radiologists.

Key Strengths in Medicine:

- State-of-the-art performance on medical imaging tasks
- Automatic feature extraction eliminates manual engineering
- Handles multiple modalities (images, text, time-series)
- Transfer learning enables use of pre-trained models

Algorithm Selection Guide

Criterion	Logistic Regression	Random Forests	SVM	Neural Networks
Data Size	Small-Medium	Medium-Large	Small-Medium	Large-Very Large
Interpretability	★★★★★	★★★	★★	★
Training Speed	Very Fast	Fast	Slow	Very Slow
Prediction Speed	Very Fast	Fast	Medium	Fast

Criterion	Logistic Regression	Random Forests	SVM	Neural Networks
Best Use Case	Risk scoring, odds ratios needed	Tabular data, feature importance	High-dimensional data, limited samples	Images, complex patterns, large datasets
Typical Accuracy	Good	Very Good	Very Good	Excellent

Regression for Biomarkers

Predicting continuous outcomes - lab values, disease progression, dosages

Linear Regression

Simple, interpretable baseline model

Use case: Predicting HbA1c levels from patient features

Ridge / Lasso / Elastic Net

Regularized regression preventing overfitting

Use case: Gene expression → biomarker prediction

Random Forest Regression

Non-linear relationships, feature importance

Use case: ICU length of stay prediction

Gradient Boosting (XGBoost)

State-of-the-art performance on tabular data

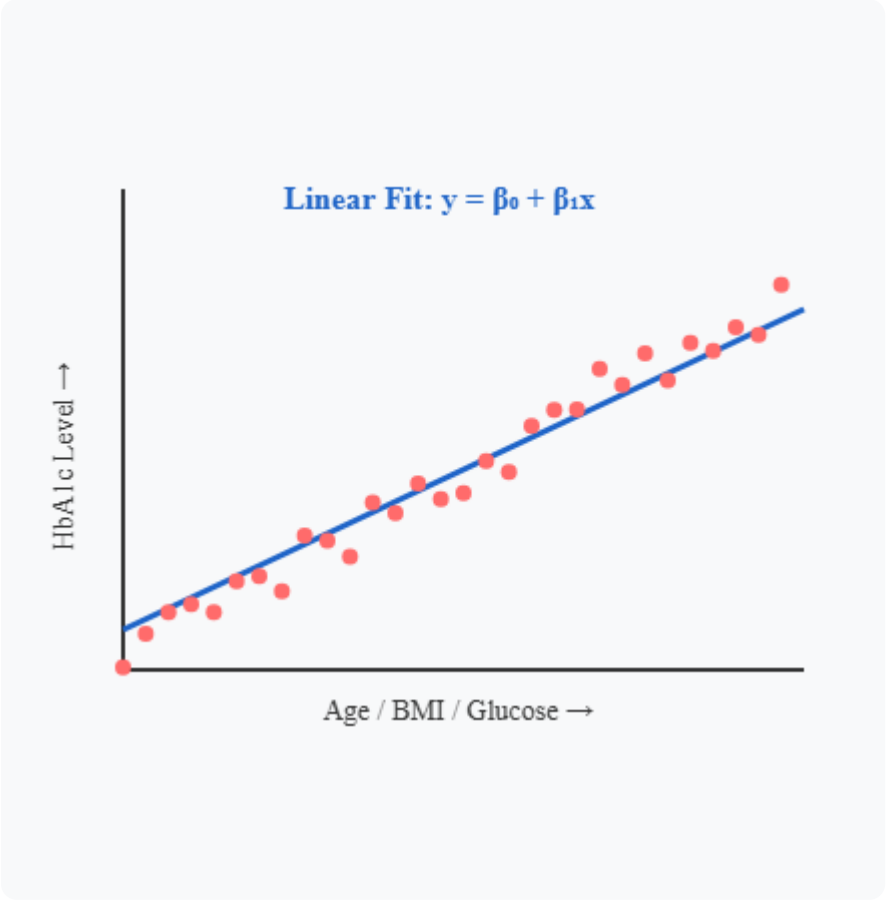
Use case: Drug dosage optimization

⚠ Critical: Prediction Intervals

Clinical decisions require not just point estimates but confidence intervals - quantify uncertainty!

Detailed Methods & Applications

1. Linear Regression



What is Linear Regression?

Linear regression models the relationship between input variables (features) and a continuous output by fitting a straight line (or hyperplane in multiple dimensions) through the data points.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \varepsilon$$

Key Characteristics

- **Interpretability:** Coefficients (β) show the direct effect of each feature
- **Assumptions:** Linearity, independence, homoscedasticity, normality
- **Training:** Minimizes Mean Squared Error (MSE)

✓ Advantages

✗ Limitations

- Highly interpretable coefficients
- Fast training and prediction
- Works well with limited data
- Provides confidence intervals easily

- Assumes linear relationships
- Sensitive to outliers
- Cannot capture complex interactions
- Struggles with high-dimensional data

📊 Clinical Example: HbA1c Prediction

Scenario: Predicting HbA1c levels based on patient age, BMI, fasting glucose, and exercise frequency.

Model: HbA1c = 4.2 + 0.03(Age) + 0.08(BMI) + 0.02(Fasting_Glucose) - 0.15(Exercise_Hours)

Interpretation: Each 1 kg/m² increase in BMI is associated with a 0.08% increase in HbA1c, while each additional hour of weekly exercise decreases HbA1c by 0.15%.

2. Ridge / Lasso / Elastic Net Regression

What is Regularized Regression?

Regularization adds a penalty term to the loss function to prevent overfitting by constraining coefficient magnitudes. This is crucial when dealing with many features or correlated predictors.

$$\text{Ridge (L2): Loss} + \lambda\sum\beta_i^2$$
$$\text{Lasso (L1): Loss} + \lambda\sum|\beta_i|$$
$$\text{Elastic Net: Loss} + \lambda_1\sum|\beta_i| + \lambda_2\sum\beta_i^2$$

Key Differences

- **Ridge:** Shrinks coefficients but keeps all features
- **Lasso:** Can reduce coefficients to exactly zero (feature selection)
- **Elastic Net:** Combines both L1 and L2 penalties

✓ Advantages

- Handles multicollinearity effectively
- Prevents overfitting with many features
- Lasso provides automatic feature selection
- Elastic Net balances both approaches

✗ Limitations

- Requires tuning regularization parameter (λ)
- Still assumes linear relationships
- Feature scaling is critical
- Interpretation becomes more complex

📊 Clinical Example: Gene Expression Biomarker Prediction

Scenario: Predicting tumor size from 5,000 gene expression levels with only 200 patient samples.

Challenge: Many more features than samples (p >> n problem) causes overfitting in standard linear regression.

Solution: Lasso regression identifies 47 genes with non-zero coefficients, providing both prediction and biological insight into which genes drive tumor growth.

Result: Test R^2 improved from 0.23 (linear) to 0.68 (Lasso) by preventing overfitting.

3. Random Forest Regression

What is Random Forest Regression?

Random Forest builds multiple decision trees on random subsets of data and features, then averages their predictions. This ensemble approach captures non-linear relationships and complex interactions.

How It Works

- **Bootstrap Sampling:** Each tree trains on a random sample with replacement
- **Random Features:** At each split, only a random subset of features is considered
- **Aggregation:** Final prediction is the average of all tree predictions
- **Feature Importance:** Calculated by measuring prediction accuracy decrease when a feature is permuted

✓ Advantages

- Captures non-linear relationships automatically
- Handles missing data well
- Provides feature importance rankings
- Robust to outliers
- No feature scaling required

✗ Limitations

- Less interpretable than linear models
- Can overfit with too many/deep trees
- Larger memory footprint
- Slower prediction than linear models

📊 Clinical Example: ICU Length of Stay Prediction

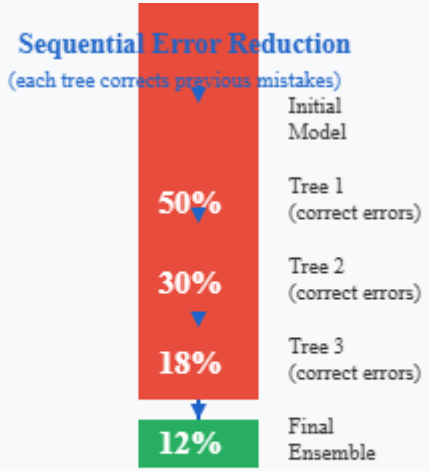
Scenario: Predicting ICU stay duration using admission vitals, lab results, comorbidities, and treatment interventions.

Complexity: Relationships are highly non-linear (e.g., U-shaped relationship between blood pressure and LOS; interactions between age and organ failure).

Model Performance: Random Forest achieved RMSE of 2.3 days vs 3.8 days for linear regression.

Feature Insights: Top predictors were APACHE score (importance: 0.24), mechanical ventilation (0.18), and sepsis presence (0.15), guiding resource allocation.

4. Gradient Boosting (XGBoost)



What is Gradient Boosting?

Gradient boosting builds trees sequentially, where each new tree corrects the errors of the previous ensemble. XGBoost is an optimized implementation with regularization and efficient algorithms.

Key Concepts

- **Sequential Learning:** Each tree focuses on the residual errors of previous trees
- **Gradient Descent:** Uses gradients to minimize loss function
- **Regularization:** L1/L2 penalties on leaf weights prevent overfitting

- **Learning Rate:** Controls contribution of each tree (shrinkage)

✓ Advantages

- State-of-the-art accuracy on tabular data
- Handles mixed data types seamlessly
- Built-in feature importance
- Efficient with missing values
- Highly customizable with many hyperparameters

✗ Limitations

- Prone to overfitting without proper tuning
- Longer training time than Random Forest
- Many hyperparameters to optimize
- Less interpretable than linear models

📊 Clinical Example: Personalized Drug Dosage Optimization

Scenario: Predicting optimal warfarin dosage based on genetic variants (CYP2C9, VKORC1), demographics, medications, and clinical factors.

Challenge: Complex gene-drug-disease interactions with non-linear dose-response curves.

XGBoost Results: Mean absolute error of 0.73 mg/day vs 1.42 mg/day for clinical algorithms, reducing bleeding/clotting events by 28%.

Model Insights: SHAP values revealed that VKORC1 genotype had the largest individual impact, but age-BMI interactions were critical for elderly patients.

🎯 Model Selection Guidelines

Start Simple: Begin with Linear Regression for interpretability.

Add Regularization: Use Ridge/Lasso/Elastic Net when you have many features or multicollinearity.

Go Non-linear: Use Random Forest when relationships are complex but interpretability is still important.

Maximize Performance: Use XGBoost for best predictive accuracy on structured data.

Always: Validate on held-out test data, calculate confidence intervals, and consider clinical interpretability alongside statistical performance.

Clustering for Disease Subtypes

Discover hidden patient subgroups with distinct characteristics

K-means

Fast, scalable clustering with predefined number of clusters

Cancer subtypes from gene expression

Hierarchical Clustering

Dendrogram-based, no need to specify K upfront

Patient stratification visualization

DBSCAN

Density-based, finds arbitrary shapes, handles outliers

Anomaly detection in clinical data

Consensus Clustering

Robust clustering through multiple runs and voting

Stable disease subtype identification

1. K-means Clustering

K-means is a partitioning method that divides data into K distinct, non-overlapping clusters. Each data point belongs to the cluster with the nearest mean (centroid). It's one of the most popular clustering algorithms due to its simplicity and efficiency.

Algorithm Steps:

1. Initialize K centroids randomly
2. Assign each point to nearest centroid
3. Recalculate centroids as cluster means
4. Repeat steps 2-3 until convergence

Key Characteristics:

- Requires specifying K (number of clusters) in advance
- Uses Euclidean distance for similarity
- Assumes spherical, equal-sized clusters
- Time complexity: O(n·K·i·d) where i = iterations, d = dimensions

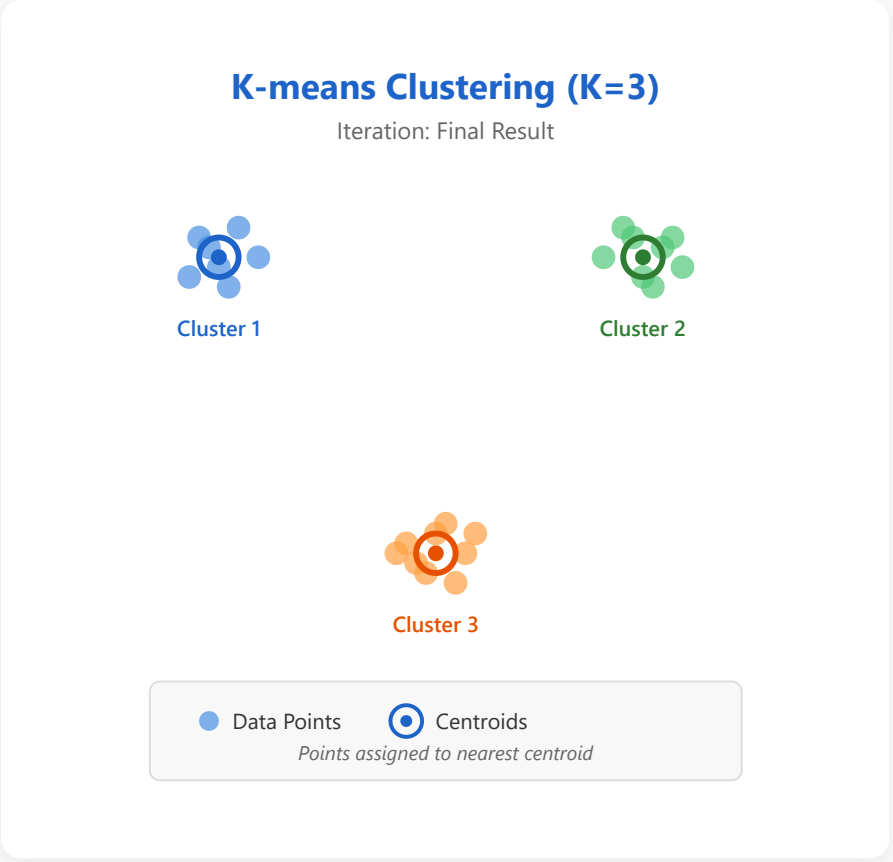
✓ Advantages

- Fast and scalable
- Easy to implement
- Works well with large datasets

✗ Limitations

- Sensitive to initialization
- Struggles with non-spherical shapes
- Affected by outliers

Clinical Example:



Identifying breast cancer subtypes (Luminal A, Luminal B, HER2+, Basal-like) based on gene expression profiles of ER, PR, and HER2 markers.

2. Hierarchical Clustering

Hierarchical clustering builds a tree-like structure (dendrogram) showing relationships between data points. It can be agglomerative (bottom-up) or divisive (top-down), with agglomerative being more common in biomedical applications.

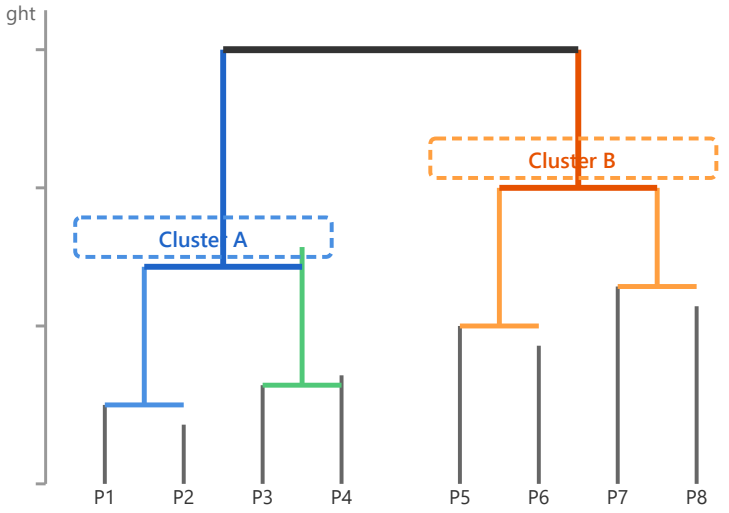
Algorithm Steps (Agglomerative):

1. Start with each point as its own cluster
2. Find the two most similar clusters
3. Merge them into a single cluster
4. Repeat until all points are in one cluster

Key Characteristics:

- No need to specify K in advance
- Creates hierarchical structure (dendrogram)
- Multiple linkage methods: single, complete, average, Ward

Hierarchical Clustering Dendrogram



Cut height determines number of clusters

- Time complexity: $O(n^3)$ or $O(n^2 \log n)$ with optimizations

Linkage Methods:

- **Single:** Minimum distance between clusters
- **Complete:** Maximum distance between clusters
- **Average:** Average distance between all pairs
- **Ward:** Minimizes within-cluster variance

✓ Advantages

- No pre-specified K needed
- Dendrogram shows relationships
- Deterministic results

✗ Limitations

- Computationally expensive
- Doesn't scale to large datasets
- Sensitive to noise/outliers

Clinical Example:

Patient stratification based on multiple clinical variables (age, lab values, symptoms) to visualize patient similarity and identify natural groupings for personalized treatment.

3. DBSCAN (Density-Based Spatial Clustering)

DBSCAN identifies clusters based on density - regions where points are closely packed together. Unlike K-means, it can find arbitrarily-shaped clusters and automatically identifies outliers as noise points.

Algorithm Steps:

1. Pick a random unvisited point
2. Find all points within ε distance (neighbors)
3. If neighbors ≥ minPts, start a new cluster
4. Recursively add density-reachable points
5. Mark isolated points as noise/outliers

Key Parameters:

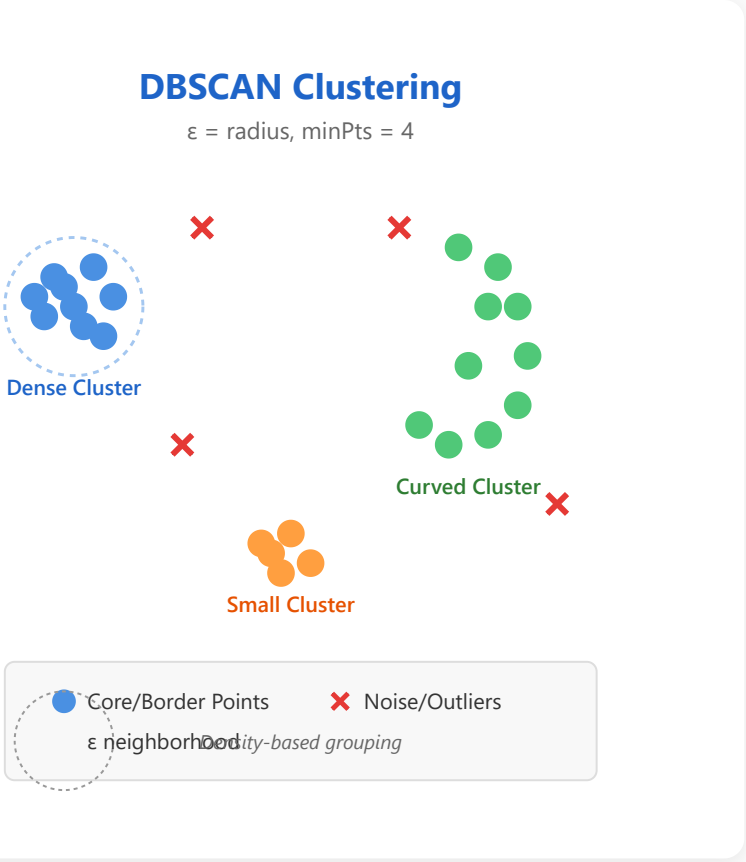
- **ε (epsilon):** Maximum distance for neighborhood
- **minPts:** Minimum points to form dense region
- **Core point:** Has ≥ minPts within ε distance
- **Border point:** In neighborhood of core point
- **Noise point:** Neither core nor border

✓ Advantages

- Finds arbitrary-shaped clusters
- Identifies outliers/noise
- No need to specify K
- Robust to outliers

✗ Limitations

- Struggles with varying densities
- Sensitive to ε and minPts
- High-dimensional challenges



Clinical Example:

Detecting unusual patient clusters in electronic health records that may represent rare disease phenotypes or adverse drug reactions, while identifying outlier cases that need special attention.

4. Consensus Clustering

Consensus clustering improves clustering stability and reliability by running multiple clustering iterations with resampled data, then combining results through voting. It provides confidence measures for cluster assignments.

Algorithm Steps:

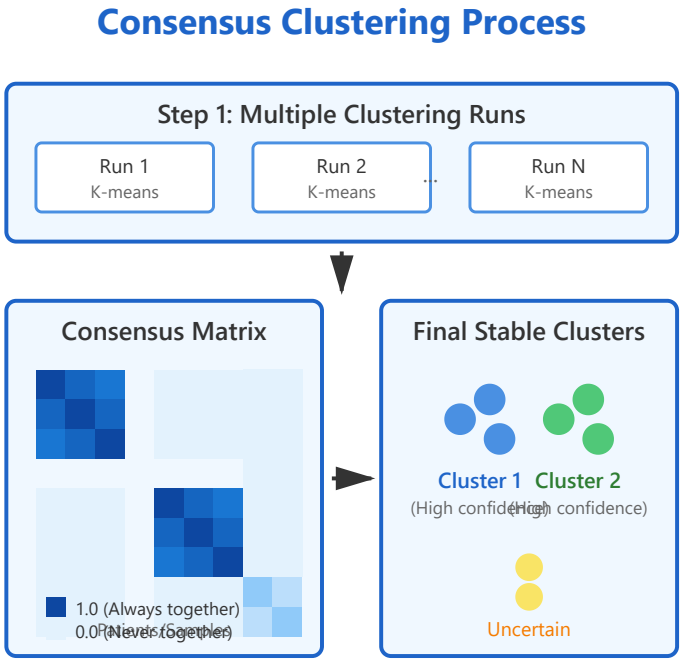
1. Resample data multiple times (e.g., 1000 runs)
2. Apply base clustering algorithm (e.g., K-means) to each sample
3. Record co-clustering frequency for all pairs
4. Build consensus matrix from frequencies
5. Apply final clustering to consensus matrix

Key Characteristics:

- Meta-clustering approach (combines multiple runs)
- Provides stability assessment via consensus matrix
- Helps determine optimal number of clusters
- Reduces sensitivity to initialization and sampling
- Can use any base clustering algorithm

Consensus Matrix:

- Entry (i,j) = proportion of runs where i and j are in same cluster
- Values near 1: strong evidence of co-clustering
- Values near 0: strong evidence of separation



Stability across multiple runs increases confidence

Typical: 100-1000 iterations with 80% resampling

- Intermediate values: uncertain assignments

✓ Advantages

- More robust and stable results
- Provides confidence measures
- Helps determine optimal K
- Reduces algorithm bias

✗ Limitations

- Computationally intensive
- Requires many iterations
- More complex to implement

Clinical Example:

Identifying stable molecular subtypes in heterogeneous cancers (e.g., glioblastoma) where robust classification is critical for treatment decisions and prognosis prediction.

Method Comparison Summary

Method	Pre-specify K?	Scalability	Cluster Shape	Best For
K-means	✓ Yes	★★★ High	Spherical	Large datasets, well-separated groups
Hierarchical	✗ No	★ Low	Any	Visualization, small-medium datasets

Method	Pre-specify K?	Scalability	Cluster Shape	Best For
DBSCAN	✗ No	⭐⭐ Medium	Arbitrary	Outlier detection, irregular shapes
Consensus	✓ Yes	⭐ Low	Depends on base	Stability, critical decisions

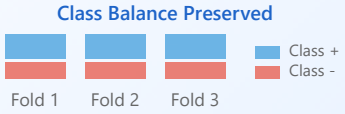
Cross-validation Strategies

K-Fold CV

Split data into K folds, train on K-1, test on 1, repeat K times

Standard: K=5 or 10

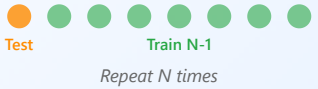
Stratified CV



Preserve class distribution in each fold - crucial for imbalanced data

Use for classification

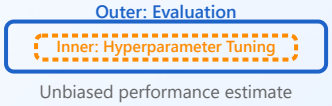
Leave-One-Out



Train on N-1 samples, test on 1, repeat N times

For very small N

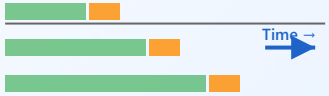
Nested CV



Outer loop for evaluation, inner loop for hyperparameter tuning

Unbiased estimates

Time Series Split



Train on past, test on future - respects temporal order

For longitudinal data

Group CV

Keep all samples from same patient/site together

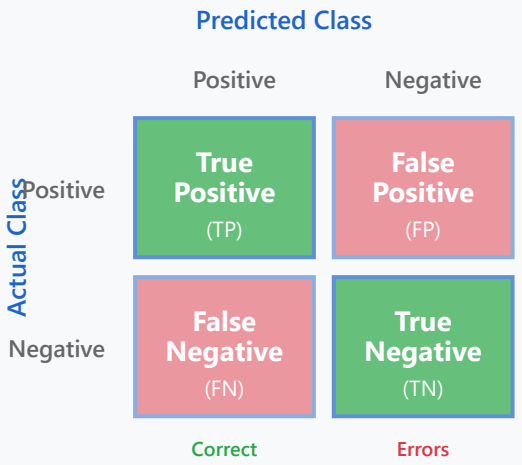
Prevent data leakage

⚠ Biomedical Pitfall

Multiple samples from same patient must stay in same fold to avoid optimistic bias!

Performance Metrics

Confusion Matrix



Sensitivity (Recall)

`TP / (TP + FN)`

How many actual positives detected

Specificity

`TN / (TN + FP)`

How many actual negatives identified

PPV (Precision)

`TP / (TP + FP)`

Positive predictive value

NPV

`TN / (TN + FN)`

Negative predictive value

F1 Score

`2 · Precision · Recall / (Precision+Recall)`

Harmonic mean of Precision & Recall

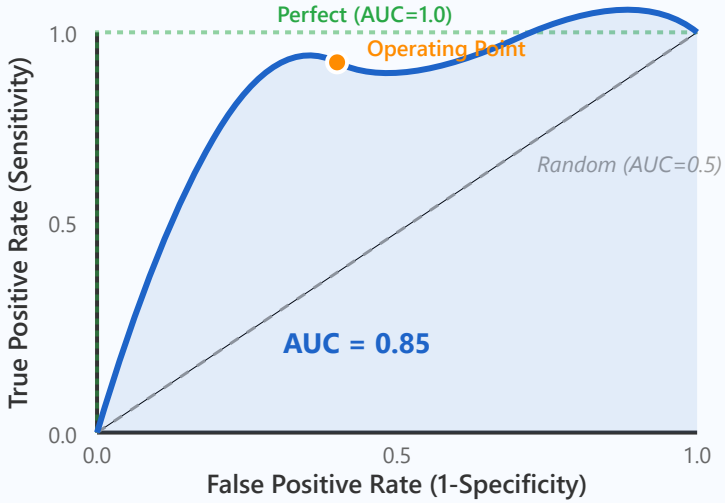
Matthews CC

`Balanced measure even for imbalanced data`

Range: -1 to +1, 0 = random

ROC and PR Curves

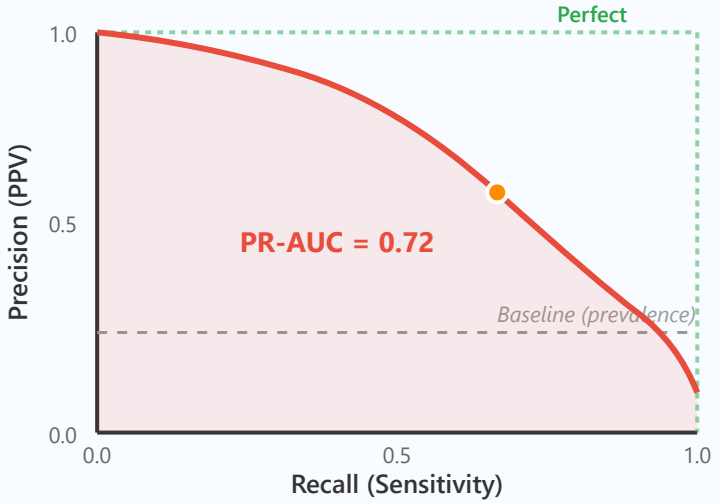
ROC Curve



Plots TPR vs FPR at various thresholds

- AUC: 0.5 = random, 1.0 = perfect
- Good for balanced datasets
- Threshold-independent metric

PR Curve



Plots Precision vs Recall

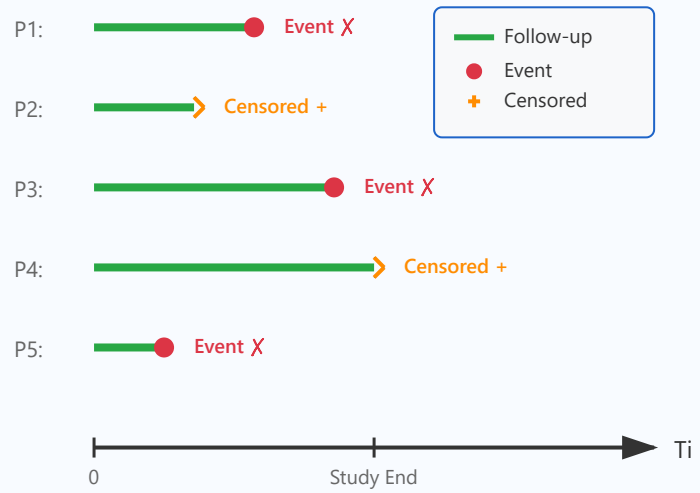
- Better for imbalanced data
- Focus on positive class
- More informative for rare diseases

Clinical Decision: Choose operating point based on cost of FP vs FN

Survival Analysis

Time-to-event analysis: Death, disease recurrence, hospital readmission

Censoring Visualization



Censoring Types

- Right censoring (most common)
- Left censoring
- Interval censoring

Key Functions

- Survival function S(t)
- Hazard function h(t)
- Cumulative hazard H(t)

Advanced Topics

- Competing risks
- Recurrent events
- Time-varying covariates

Clinical Applications

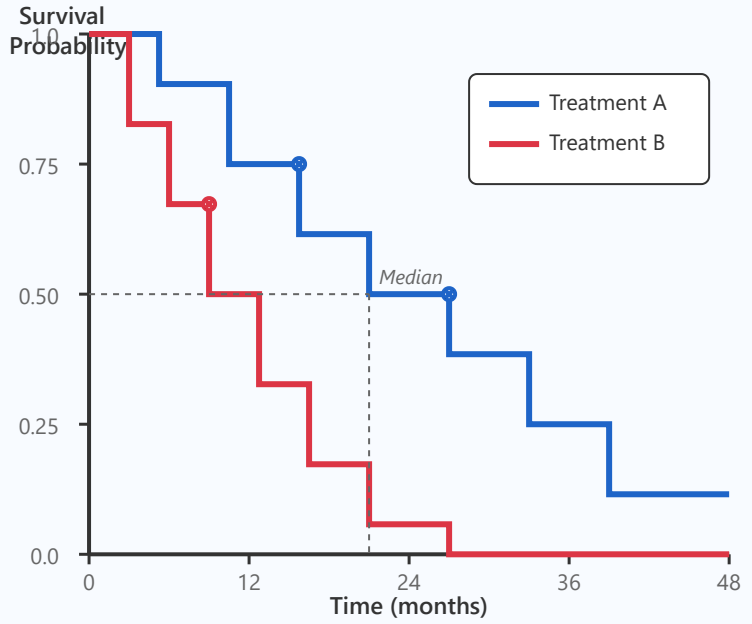
- Overall survival (OS)
- Progression-free survival (PFS)
- Time to treatment failure

Kaplan-Meier Survival Curve

Survival Curve Example

Kaplan-Meier Method

A **non-parametric survival estimation** method that accurately calculates survival probabilities even with censored data.



$$S(t) = \prod(1 - d_i/n_i)$$

- d_i: Number of events at time i
- n_i: Number at risk at time i
- Represented as a step function

Interpretation Points

- **Median survival**: Time when S(t) = 0.5
- **Curve comparison**: Use log-rank test
- **Confidence intervals**: Greenwood's formula
- Treatment A shows superior survival vs B

Cox Proportional Hazards Model

Hazard Ratio Visualization

Cox Proportional Hazards Model

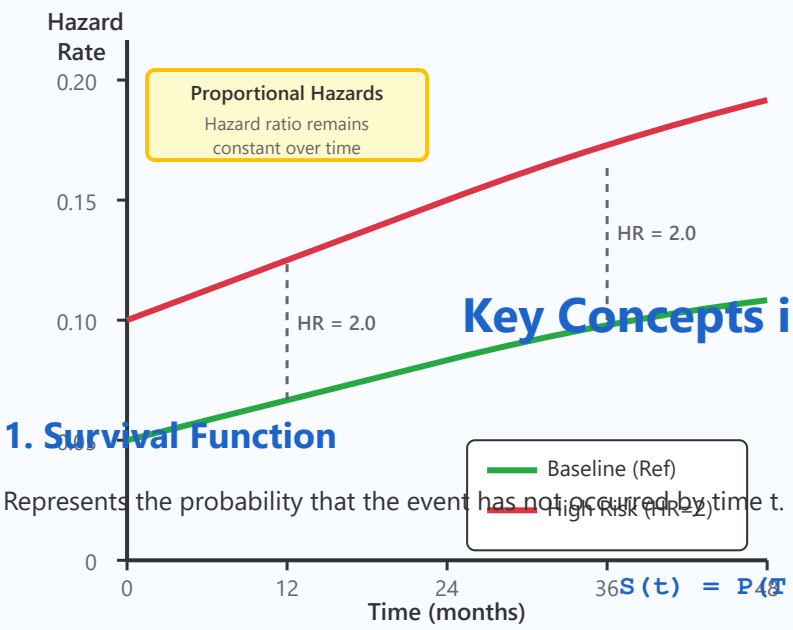
A **semi-parametric regression model** that simultaneously evaluates the effects of multiple covariates on survival.

$$h(t|X) = h_0(t) \times \exp(\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p)$$

- $h_0(t)$: Baseline hazard function
- **exp(β)**: Hazard Ratio (HR)
- **Proportionality assumption**: HR is time-independent

Hazard Ratio (HR) Interpretation

Key Concepts in Survival Analysis



1. Survival Function

Represents the probability that the event has not occurred by time t.

$$S(t) = P(T > t) = 1 - F(t)$$

- S(0) = 1 (all subjects survive at start)
- S(∞) = 0 (eventually all experience event)
- Monotonically decreasing function

2. Hazard Function

The instantaneous risk of experiencing the event at time t, given survival until t.

$$h(t) = \lim_{\Delta t \to 0} P(t \leq T < t+\Delta t \mid T \geq t) / \Delta t$$

- **Constant**: Exponential distribution
- **Increasing**: Weibull distribution
- **Decreasing then increasing**: Log-normal

3. Cumulative Hazard

The accumulated hazard from time 0 to time t.

$$H(t) = \int_0^t h(u)du = -\ln[S(t)]$$

- Estimated by Nelson-Aalen estimator
- S(t) = exp[-H(t)]
- Interconvertible with survival function

4. Log-Rank Test

- Schoenfeld residuals test
- Log-log survival curve parallelism check

Clinical Research Application Example

Study Scenario: New Drug Clinical Trial

5. Censoring Handling

Objective: Evaluate whether a new cancer drug (Drug A) improves overall survival compared to standard treatment.

Population: 200 patients with advanced lung cancer (100 per arm)

Follow-up: Maximum 48 months

$$\chi^2 = [\Sigma(O_i - E_i)]^2 / \Sigma \text{Var}(O_i - E_i)$$

- Compares observed vs expected events

Step 1: Study Design

- **Primary endpoint**: Overall survival (OS)
- **Secondary endpoints**: Progression-free survival (PFS)
- **Covariates**: Age, sex, disease stage, ECOG score
- **Event definition**: Death from any cause

Step 2: Data Collection

Methods to utilize information from incomplete observations.

- **Non-informative censoring**: Independent of event
- Information until censoring included
- Removed from risk set afterwards
- Key strength of survival analysis

- From enrollment to death or last follow-up
- Censored: Lost to follow-up, withdrew, study end
- Events observed: 118 (59%)
- Censored: 82 (41%)

Step 3: Statistical Analysis

1 Kaplan-Meier Analysis

- Estimate survival curves for each arm
- Calculate median survival times
- Present 95% confidence intervals

Results:

Drug A: Median survival 26.5 months

Standard: Median survival 18.2 months

2 Log-Rank Test

- Compare survival curves between arms
- Consider entire follow-up period

Results:

$\chi^2 = 8.42$, $p = 0.004$

→ Statistically significant difference

Advanced Topics and Considerations

Violation of Proportional Hazards

Alternatives when the key Cox model assumption is not met:

1. Stratified Cox Model

- Stratify by variables violating proportionality
- Estimate separate baseline hazards per stratum

2. Time-Dependent Covariates

- Include time-varying variables
- Example: X(t) = X × g(t)

3. Accelerated Failure Time Model

Competing Risks Analysis

When multiple types of events can occur:

Example Scenarios

- Cancer death vs other-cause death
- Recurrence vs death
- Transplant: Transplanted vs died waiting

Analysis Methods

- **Cumulative Incidence Function (CIF)**: Probability of specific event
- **Fine-Gray Model**: Subdistribution hazard
- Kaplan-Meier may overestimate with competing risks

• Directly model survival time
• Use log-normal, Weibull distributions, etc.

ndar

Treatment effect remains significant after adjusting for other prognostic factors

Recurrent Events Analysis

Multiple events possible in same subject:

• Hospitalizations, recurrences, infections, etc.

Analysis Approaches

• **Anderson-Gill Model**: All events independent
• **PWP Model**: Sequential conditional events
• **Frailty Model**: Account for within-subject correlation

Sample Size Calculation

Considerations for survival study design:

$$\text{Events} = (Z_{1-\alpha/2} + Z_{1-\beta})^2 / [p_1(1-p_1)(\log HR)^2]$$

• Number of events determines power
• Consider follow-up duration and dropout
• Higher censoring requires more enrollment

Key Cautions and Checklist

⚠ Data Quality

✓ Is event definition clear?
✓ Are censoring reasons documented?
✓ Is the time origin clearly defined?

⚠ Result Reporting

✓ Present median follow-up time
✓ Display number at risk
✓ Report with confidence intervals

⚠ Model Assumptions

✓ Test proportional hazards assumption
✓ Verify non-informative censoring
✓ Check for outliers and influential observations

⚠ Interpretation

✓ Distinguish clinical vs statistical significance
✓ Consider adjustment for multiple testing
✓ Check for potential confounders

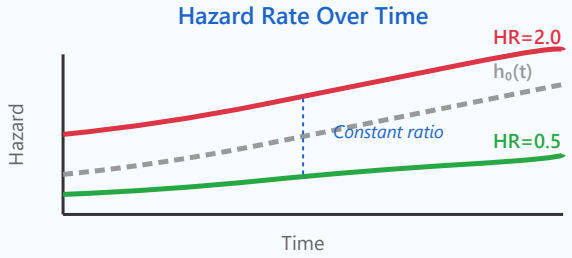
Cox Proportional Hazards Model

Gold standard for survival analysis in clinical research

Model Formula

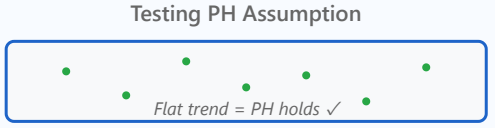
$$h(t|X) = h_0(t) \cdot \exp(\beta_1 X_1 + \ldots + \beta_p X_p)$$

Hazard ratio = exp(β)



Key Assumptions

- **Proportional hazards:** HR constant over time
- **Linear relationship:** with log-hazard
- **Independent censoring:** uninformative



Time-varying Covariates
Handle changing predictors

Stratification
When PH assumption violated

Penalized Cox
High-dimensional data

🔍 Principles and Detailed Explanation of Cox Regression

📊 Semi-parametric Approach

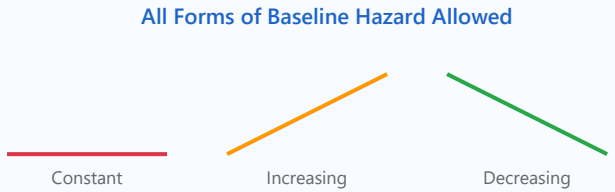
The Cox model is a **semi-parametric** method that does not assume a specific form for the baseline hazard function $h_0(t)$.

⚖️ Partial Likelihood Estimation

The Cox model estimates β using **partial likelihood**.

At each event time, it compares "who has a higher probability of experiencing the event?"

Advantage: It can estimate the effect of covariates without needing to know the exact distribution of hazard over time.



$$L(\beta) = \prod_i \left[\exp(\beta X_i) / \sum_{j \in R_i} \exp(\beta X_j) \right]$$

R_i = risk set at time i

Cox Model Fitting Process

STEP 1: Data Preparation

- Verify survival time, event indicator, and covariates
- Identify censored observations (event=0)

STEP 2: Construct Risk Sets

- At each event time, create a set of all subjects who have not yet experienced the event
- Example: Event at t=5 → All subjects surviving up to time 5 form the risk set

STEP 3: Calculate Partial Likelihood

- Compute the relative hazard of the subject who actually experienced the event at each time point
- Baseline hazard $h_0(t)$ cancels out in calculations (only ratios matter)

STEP 4: Maximize and Estimate β

- Use Newton-Raphson algorithm to maximize partial likelihood
- Estimate regression coefficients β and standard errors

STEP 5: Assumption Testing and Model Diagnostics

- Test proportional hazards assumption (Schoenfeld residuals test)
- Check for influential observations (dfbeta, score residuals)

Hazard Ratio (HR) Interpretation Guide

HR = exp(β) represents the change in hazard when a covariate increases by one unit.

HR = 2.0	HR = 1.0	HR = 0.5
2x increase in risk (100% ↑) e.g., Smoker vs Non-smoker	No effect (0% change) Null hypothesis state	50% reduction in risk (halved ↓) e.g., Treatment effect

📋 Real Research Example: Cancer Patient Survival Analysis

Study Setting: 200 cancer patients, 5-year follow-up
Variables: Age, tumor size, treatment type
Results:
• Age: β=0.03, HR=1.03 (3% increase in risk per 1-year increase)
• Tumor size: β=0.25, HR=1.28 (28% increase in risk per 1cm increase)
• New treatment: β=-0.51, HR=0.60 (40% reduction in mortality risk vs. standard, $p<0.001$)

Interpretation: The new treatment significantly improves survival even after adjusting for age and tumor size.

✅ When to Use Cox Model

- Survival analysis with censored data
- When the exact distribution of hazard over time is unknown
- To quantify the effect of covariates on risk
- When the proportional hazards assumption is valid (or can be appropriately transformed)

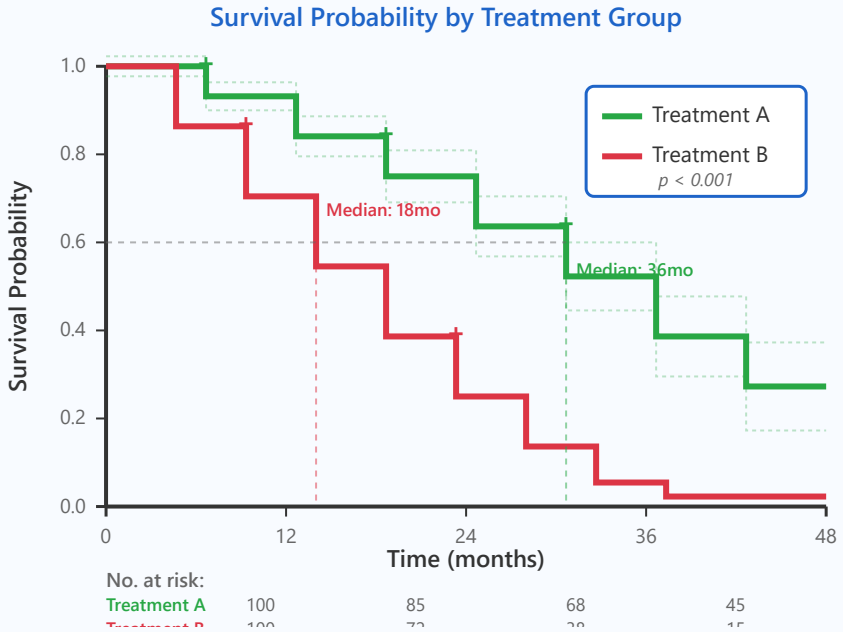
⚠️ Cautions

- Interpretation requires caution when proportional hazards assumption is violated
- Estimation becomes unstable with small sample size or few events
- Time-dependent covariates cannot be handled by standard Cox model
- Competing risks require specialized models

Kaplan-Meier Survival Curves

Non-parametric estimator of survival function

KM Curve Example



Key Components

- Step function visualization
- Confidence intervals (95% CI)
- Number at risk table
- Censoring markers (+)
- Median survival time

Log-rank Test

- Compare survival curves
- Null: no difference in survival
- P-value < 0.05 = significant
- Non-parametric test
- Assumption: PH holds

Clinical Interpretation: Always report median survival, 95% CI, and number at risk at key timepoints

Kaplan-Meier Estimation Principle

KM Estimation Formula

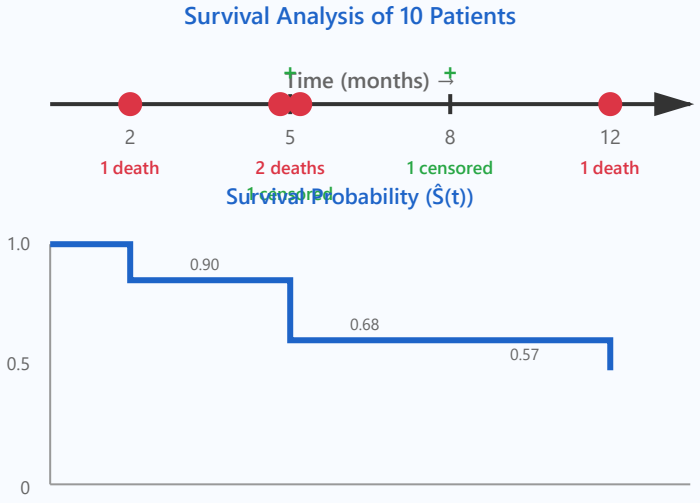
$$\hat{S}(t) = \prod_{t_i \le t} (1 - d_i/n_i)$$

Variable Definitions:

- **t_i**: Time of event occurrence
- **d_i**: Number of events at t_i
- **n_i**: Number at risk just before t_i
- **$\hat{S}(t)$**: Estimated survival probability at time t

- Calculate conditional survival probability at each event time
- Multiply all previous conditional probabilities (product-limit)
- Censored data only excluded from n_i, not counted as events

Calculation Example Visualization



Step-by-Step Calculation Process

Time (t_i)	At Risk (n_i)	Deaths (d_i)	Censored (c_i)	Conditional Survival ($1 - d_i/n_i$)	Cumulative Survival $\hat{S}(t_i)$
0	10	0	0	1.000	**1.000**
2 months	10	1	0	9/10 = 0.900	**1.000 × 0.900 = 0.900**
5 months	9	2	1	7/9 = 0.778	**0.900 × 0.778 = 0.700**
8 months	6	0	1	6/6 = 1.000	**0.700 × 1.000 = 0.700**
12 months	5	1	0	4/5 = 0.800	**0.700 × 0.800 = 0.560**

Key Points:

- Censored patients are excluded from the at-risk group after that time point, but not counted as events
- Calculate cumulative survival probability by multiplying conditional survival probabilities at each time point (Product-Limit Estimator)
- Survival probability does not change at time points without events (Step function)
- Survival probability beyond the last observation time cannot be estimated

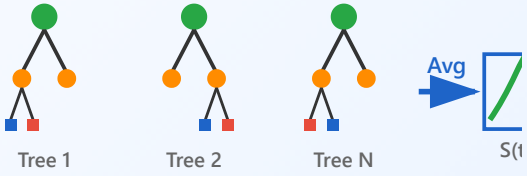
Important: The KM curve is a non-parametric method that estimates the survival function without distribution assumptions and has the advantage of properly handling censored data.

Time-to-Event Prediction with Machine Learning

Comprehensive Guide to Modern Survival Analysis Methods

Table of Contents

Survival analysis, also known as time-to-event analysis, focuses on predicting the time until an event of interest occurs. Machine learning has revolutionized this field by providing powerful methods that can handle complex, non-linear relationships and high-dimensional data.



Ensemble of survival trees
Bootstrap aggregation for robust predictions

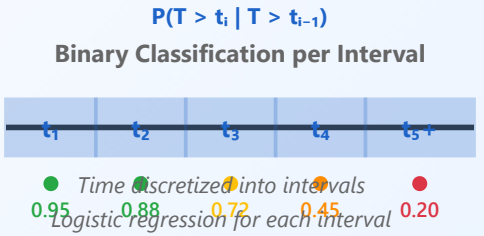
Random Survival Forests

Ensemble learning approach that combines multiple survival trees. Handles non-linear relationships and provides feature importance rankings.

Deep neural network architecture
Cox partial likelihood loss function

DeepSurv

Deep learning approach combining neural networks with Cox regression. Captures complex non-linear patterns in high-dimensional data.



Time discretized into intervals
Logistic regression for each interval

Discrete Time Models

Transforms survival analysis into sequence of binary classification problems. Flexible and easy to implement with standard ML tools.

Evaluation Metrics for Survival Analysis

- C-index (Concordance)
- Time-dependent AUC
- Calibration Plots
- Brier Score
- Integrated Brier Score

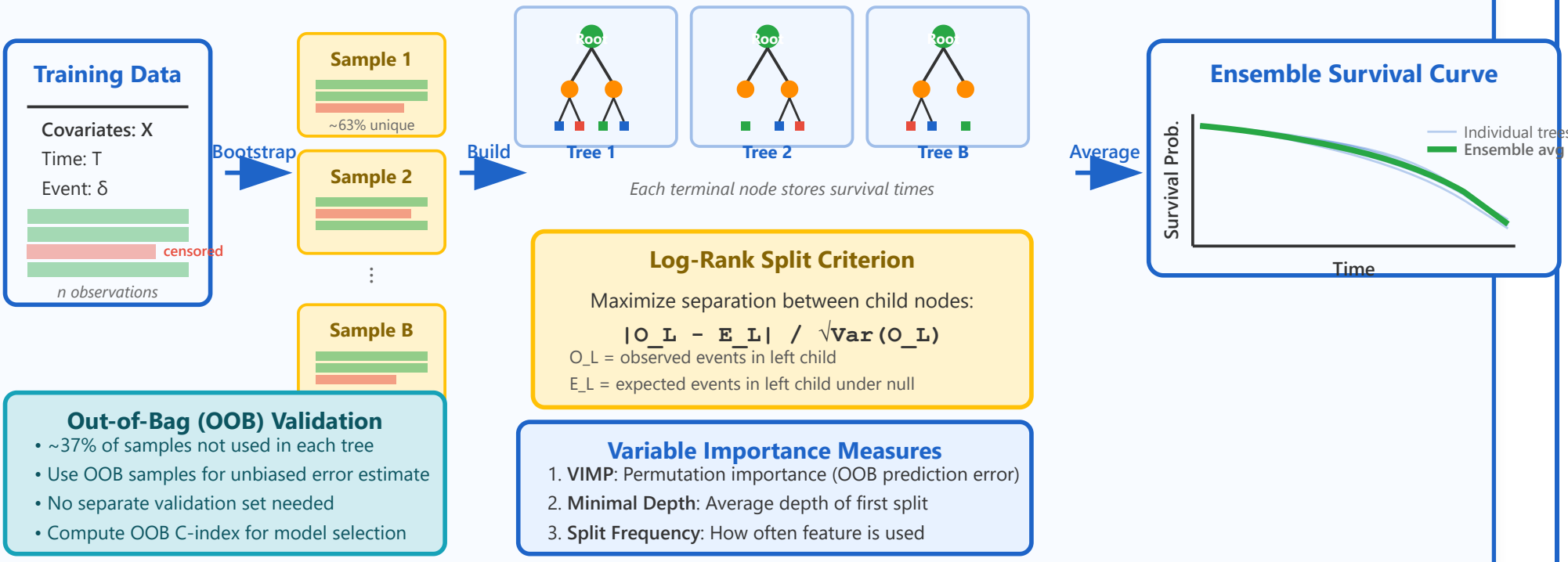
METHOD 1

Random Survival Forests (RSF)

Overview & Core Concept

Random Survival Forests extend the traditional Random Forest algorithm to handle censored survival data. Instead of predicting a class label or continuous value, RSF estimates the entire survival function S(t) for each individual, representing the probability of surviving beyond time t. The method builds an ensemble of survival trees, each trained on a bootstrap sample of the data, and aggregates their predictions to produce robust, non-parametric survival estimates that can capture complex patterns without assuming proportional hazards or specific distributional forms.

Complete RSF Architecture: From Data to Prediction



Complete workflow of Random Survival Forests: Data is bootstrap sampled, survival trees are grown using log-rank splitting, predictions from all trees are averaged to create ensemble survival function. OOB samples provide unbiased validation, and feature importance can be extracted.

▸ Key Features & Advantages

🌳 Non-parametric & Flexible

Makes no assumptions about the baseline hazard or proportional hazards. Can model complex, non-linear relationships between

🔍 Automatic Feature Selection

Automatically identifies important predictors through variable importance measures (VIMP, minimal depth). Robust to irrelevant features and multicollinearity.

covariates and survival times, including interactions that would be difficult to specify manually.

💔 Handles Missing Data

Can use surrogate splits when features have missing values, maintaining prediction accuracy even with incomplete data. No need for imputation in many cases.

📊 Built-in Cross-validation

Out-of-bag (OOB) samples (~37% per tree) provide unbiased estimate of prediction error without requiring a separate validation set. Efficient use of data.

⚡ Parallelizable Training

Trees are grown independently, allowing for parallel computation across multiple cores/machines. Scales well to large datasets.

🎯 Robust Predictions

Ensemble averaging reduces variance and overfitting. More stable predictions compared to single trees, especially with complex or noisy data.

▸ Algorithm Details & Implementation Steps

The RSF algorithm follows these key steps for building each survival tree in the forest:

- **Bootstrap Sampling:** Draw a random sample with replacement from the training data. Typically results in about 63% unique observations per tree, with remaining 37% as out-of-bag (OOB) samples for validation.
- **Random Feature Selection:** At each node during tree growing, randomly select a subset of m features from all p available features. Common choices: m = √p for survival data or m = p/3 for regression-type problems.
- **Split Point Selection:** For each selected feature, find the optimal split point that maximizes survival difference between daughter nodes using the log-rank test statistic. The split maximizes |O_L - E_L| / √Var(O_L), where O_L is observed events and E_L is expected events in the

left child.

- **Recursive Partitioning:** Recursively apply splitting until a stopping criterion is met: minimum node size reached (e.g., 3-15 observations), no significant splits available (p-value threshold), or maximum depth achieved.
- **Terminal Node Estimation:** In each leaf node, estimate the survival function using the Kaplan-Meier estimator based on the training samples that fall into that node. This gives $S_{leaf}(t)$ for all time points.
- **Ensemble Prediction:** For a new observation x, drop it down each of the B trees to obtain B terminal nodes. Collect the B survival function estimates and average them: $S_{ensemble}(t|x) = (1/B) \Sigma S_b(t|x)$.
- **Variable Importance:** Calculate feature importance using permutation: for each feature, randomly permute its values in OOB samples and measure decrease in prediction accuracy (C-index). Larger decreases indicate more important features.

⚡ Hyperparameter Tuning Guide

Number of trees (ntree): 500-1000 typically sufficient. More trees = more stable predictions but longer training. Error plateaus after sufficient trees.

Features per split (mtry): $\sqrt{p}$ is default and works well. Try $p/3$ or $\log_2(p)$ for alternatives. Smaller values = more diversity, larger values = stronger individual trees.

Minimum node size (nodesize): 3-15 typically. Larger values = simpler trees, less overfitting, faster training. Smaller values = more complex trees, potential overfitting.

Splitting rule: Log-rank (default), log-rank score, or conservation of events. Log-rank typically performs best for standard survival analysis.

Computational cost: Training time is $O(ntree \times n \times p \times \log n)$. Use parallel processing with multiple cores. Prediction is fast: $O(ntree \times \log n)$ per observation.

▸ Practical Example & Code Implementation

Clinical Application: Predicting cancer patient survival after diagnosis using clinical features (age, tumor stage, biomarkers), treatment information, and patient characteristics. RSF can identify complex interactions between age, tumor characteristics, and treatment response that proportional hazards models might miss.

```
# Python Implementation using scikit-survival
from sksurv.ensemble import RandomSurvivalForest
from sksurv.metrics import concordance_index_censored
import numpy as np
import matplotlib.pyplot as plt

# Initialize Random Survival Forest
rsf = RandomSurvivalForest(
    n_estimators=1000,          # Number of trees
    min_samples_split=10,       # Min samples to split node
    min_samples_leaf=15,        # Min samples in leaf
    max_features="sqrt",        # Features to consider per split
    n_jobs=-1,                  # Use all CPU cores
    random_state=42
)

# Train model
# y_train is structured array with fields 'event' (bool) and 'time' (float)
rsf.fit(X_train, y_train)

# Get survival function for new patients
surv_funcs = rsf.predict_survival_function(X_test)

# Plot survival curves for first 5 patients
```

```
for i, surv_func in enumerate(surv_funcs[:5]):
    plt.step(surv_func.x, surv_func.y, where="post",
             label=f"Patient {i+1}")
plt.xlabel("Time")
plt.ylabel("Survival Probability")
plt.title("Predicted Survival Curves")
plt.legend()
plt.show()

# Extract risk scores (higher = worse prognosis)
risk_scores = rsf.predict(X_test)

# Calculate C-index (concordance)
c_index = rsf.score(X_test, y_test)
print(f"C-index: {c_index:.3f}")

# Get feature importance
feature_importance = rsf.feature_importances_
important_features = np.argsort(feature_importance)[::-1][:10]
print("Top 10 important features:", important_features)
```

📚 When to Use Random Survival Forests

Ideal scenarios:

- Non-linear relationships between predictors and survival
- Need for variable importance/feature selection
- Moderate sample size (n > 100) with many features
- Missing data in predictor variables
- No strong prior belief in proportional hazards
- Want interpretable feature rankings

Less ideal for:

- Very small samples (n < 50)
- Need for explicit hazard ratios or covariate effects
- Real-time predictions with limited compute resources
- When linear Cox model performs adequately

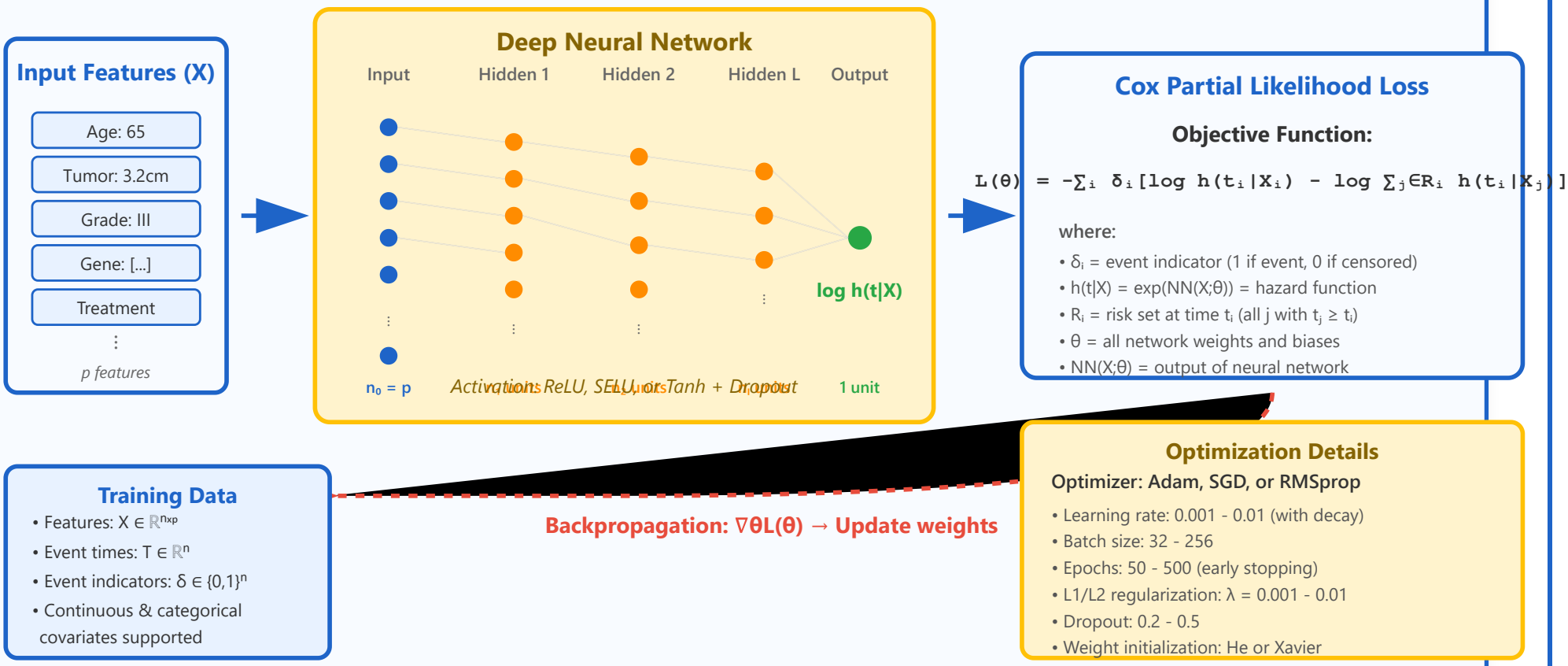
METHOD 2

DeepSurv: Deep Learning for Survival Analysis

Overview & Core Concept

DeepSurv is a deep neural network approach that extends the Cox proportional hazards model using deep learning architectures. Instead of assuming linear relationships between covariates and log-hazard like traditional Cox regression, DeepSurv uses a multi-layer feed-forward neural network to learn highly complex, non-linear representations of the input features. The network is trained using the Cox partial likelihood loss function, which maintains the semi-parametric nature and relative risk interpretation of Cox models while allowing the model to capture intricate patterns in high-dimensional data. This makes DeepSurv particularly powerful for genomic data, medical imaging features, or any scenario where relationships between predictors and survival are highly non-linear.

DeepSurv Architecture & Training Process



DeepSurv architecture showing input features processed through multiple hidden layers with non-linear activations, producing log-hazard predictions. The Cox partial likelihood loss enables end-to-end training via backpropagation while maintaining the interpretable hazard ratio framework.

▸ Key Features & Advantages

🧠 Deep Non-linear Representations

📈 High-dimensional Data Excellence

Multiple hidden layers can learn hierarchical, highly complex non-linear relationships between covariates and survival that shallow models cannot capture. Particularly powerful for discovering subtle patterns in high-dimensional data.

Excels with genomic data, medical imaging features, or other settings where p >> n (features greatly exceed samples). Deep architectures can perform implicit dimensionality reduction and feature learning.

🖇️ Automatic Interaction Learning

Automatically discovers complex interactions between features without manual specification. Hidden layers naturally combine features in non-linear ways that would be intractable to enumerate manually.

⚖️ Maintains Cox Interpretability

Uses Cox partial likelihood, so hazard ratios between patients can still be computed for risk stratification. Predictions have clear clinical interpretation as relative risks.

🎯 Transfer Learning Potential

Pre-trained networks from related tasks can be fine-tuned, leveraging knowledge from larger datasets. Particularly useful when labeled survival data is limited.

🔧 Flexible Architecture

Can incorporate various input types (continuous, categorical, images, text) with appropriate preprocessing layers. Architecture can be customized for domain-specific requirements.

▸ Network Architecture Design

Typical architecture components for DeepSurv:

- Input Layer: Dimensionality equals number of features (p). Common preprocessing: standardization (zero mean, unit variance) or normalization (min-max scaling to [0,1]). One-hot encoding for categorical variables.

- **Hidden Layers:** Usually 1-4 hidden layers with decreasing width pattern (e.g., 128 → 64 → 32 or 256 → 128 → 64). Deeper networks for very complex patterns, shallower for moderate complexity. Each layer has fewer neurons than previous to create information bottleneck.
- **Activation Functions:** ReLU (most common, fast, avoids vanishing gradients), SELU (self-normalizing, good for deep networks), or Tanh (bounded output). Applied element-wise after each hidden layer's linear transformation.
- **Dropout Layers:** Regularization via random neuron deactivation during training. Drop rate typically 0.1-0.5. Higher dropout for smaller datasets. Reduces overfitting by preventing co-adaptation of neurons.
- **Batch Normalization:** Normalizes layer inputs to have zero mean and unit variance. Stabilizes training, allows higher learning rates, acts as regularizer. Applied before or after activation.
- **Output Layer:** Single neuron with linear activation producing log-hazard ratio log h(t|X). No softmax or sigmoid needed since we're not doing classification. Output can be any real number.

```
# Example architecture specification
Layer 1 (Input):         p neurons  (features)
Layer 2 (Hidden 1):      128 neurons + ReLU + Dropout(0.3) + BatchNorm
Layer 3 (Hidden 2):      64 neurons  + ReLU + Dropout(0.3) + BatchNorm
Layer 4 (Hidden 3):      32 neurons  + ReLU + Dropout(0.2)
Layer 5 (Output):        1 neuron    (linear activation)

Forward pass:
  z1 = X
  z2 = ReLU(BatchNorm(W1·z1 + b1))  # Apply dropout in training
  z3 = ReLU(BatchNorm(W2·z2 + b2))
  z4 = ReLU(W3·z3 + b3)
  output = W4·z4 + b4  # log hazard
```

▸ Training Process & Optimization

DeepSurv is trained using mini-batch stochastic gradient descent with the Cox partial likelihood loss:

- **Batch Formation:** Divide training data into mini-batches of size B (typically 32-256). Each batch should ideally contain a mix of events and censored observations.
- **Forward Propagation:** Pass batch through network to compute log-hazards: $\hat{y}_i = NN(X_i; \theta)$ for each sample i in batch.
- **Loss Computation:** Calculate negative Cox partial log-likelihood on batch. For each observed event at time t_i with $\delta_i=1$, compute log likelihood contribution: $\hat{y}_i - \log(\sum_{j \in R_i} \exp(\hat{y}_j))$ where R_i is the risk set.
- **Backpropagation:** Compute gradients $\nabla\theta\ L(\theta)$ via automatic differentiation (chain rule applied through all layers). Modern frameworks (PyTorch, TensorFlow) handle this automatically.
- **Weight Update:** Update parameters using optimizer (Adam recommended): $\theta \leftarrow \theta - \alpha \cdot \nabla\theta L(\theta)$ where α is learning rate. Adam adapts learning rate per parameter based on gradient history.
- **Iteration:** Repeat for all batches (one epoch), then repeat for multiple epochs (50-500 typical) until convergence or early stopping triggered.

⚡ Training Best Practices

Learning Rate Strategy: Start with 0.001 (Adam default). Use ReduceLROnPlateau to decrease by factor of 10 when validation loss plateaus for 10+ epochs. Or use cyclic learning rates for better convergence.

Early Stopping: Monitor validation C-index every epoch. Stop if no improvement for 20-50 consecutive epochs. Save model with best validation performance, not the final epoch.

Regularization Balance: Start with moderate dropout (0.3) and L2 penalty (λ=0.01). If overfitting, increase. If underfitting, decrease. Can also use

L1 for feature selection.

Batch Size Selection: Larger batches (128-256) give more stable gradients but less regularization. Smaller batches (32-64) add noise that can help escape local minima. GPU memory constrains maximum size.

Initialization Matters: Use He initialization for ReLU (variance scales with $2/n_{in}$) or Xavier/Glorot for Tanh/Sigmoid. Poor initialization can cause vanishing/exploding gradients.

Validation Strategy: Use k-fold cross-validation or hold-out validation set. Never tune on test set. Monitor both C-index and calibration during training.

Implementation Example

```
# PyTorch DeepSurv Implementation
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.preprocessing import StandardScaler

class DeepSurv(nn.Module):
    def __init__(self, in_features, layers=[128, 64, 32], dropout=0.3):
        super(DeepSurv, self).__init__()

        # Build network layers
        modules = []
        prev_size = in_features

        for i, layer_size in enumerate(layers):
            # Fully connected layer
```

```
            modules.append(nn.Linear(prev_size, layer_size))
            # Batch normalization
            modules.append(nn.BatchNorm1d(layer_size))
            # Activation
            modules.append(nn.ReLU())
            # Dropout (less in final layers)
            drop_rate = dropout if i < len(layers)-1 else dropout*0.5
            modules.append(nn.Dropout(drop_rate))
            prev_size = layer_size

        # Output layer (log hazard)
        modules.append(nn.Linear(prev_size, 1))

        self.network = nn.Sequential(*modules)

    def forward(self, x):
        return self.network(x)

# Cox Partial Likelihood Loss
def cox_loss(log_hazards, times, events):
    # Sort by time (descending for risk set computation)
    idx = torch.argsort(times, descending=True)
    log_hazards = log_hazards[idx]
    events = events[idx]

    # Compute log risk = log(sum of exp(log_hazard) for risk set)
    # Use logsumexp for numerical stability
    hazard_ratio = torch.exp(log_hazards)
    log_risk = torch.log(torch.cumsum(hazard_ratio, dim=0))

    # Negative log partial likelihood (only for observed events)
    uncensored_likelihood = log_hazards - log_risk
    loss = -torch.sum(uncensored_likelihood * events) / torch.sum(events)
```

```
    return loss

# Training setup
model = DeepSurv(in_features=100, layers=[128, 64, 32], dropout=0.3)
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.01)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, patience=10)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Training loop
best_c_index = 0
patience_counter = 0

for epoch in range(500):
    model.train()
    epoch_loss = 0

    # Mini-batch training
    for batch_idx in range(0, len(X_train_scaled), batch_size):
        batch_X = torch.FloatTensor(
            X_train_scaled[batch_idx:batch_idx+batch_size])
        batch_times = torch.FloatTensor(
            times_train[batch_idx:batch_idx+batch_size])
        batch_events = torch.FloatTensor(
            events_train[batch_idx:batch_idx+batch_size])

        optimizer.zero_grad()
        log_hazards = model(batch_X)
        loss = cox_loss(log_hazards, batch_times, batch_events)
```

```
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()

    # Validation
    model.eval()
    with torch.no_grad():
        val_log_hazards = model(torch.FloatTensor(X_val_scaled))
        c_index = concordance_index(
            times_val, -val_log_hazards.numpy(), events_val)

    scheduler.step(epoch_loss)

    if c_index > best_c_index:
        best_c_index = c_index
        torch.save(model.state_dict(), 'best_model.pth')
        patience_counter = 0
    else:
        patience_counter += 1

    if patience_counter >= 30:
        print(f"Early stopping at epoch {epoch}")
        break

    if epoch % 10 == 0:
        print(f"Epoch {epoch}, Loss: {epoch_loss:.4f}, "
              f"Val C-index: {c_index:.4f}")

# Load best model for final predictions
model.load_state_dict(torch.load('best_model.pth'))
model.eval()
```

When to Use DeepSurv

Ideal scenarios:

- High-dimensional data (p >> n): genomics, imaging features
- Highly non-linear relationships expected
- Complex feature interactions present
- Sufficient training data available (n > 500 ideally)
- Computational resources available (GPU helpful)
- Want to leverage transfer learning from related domains

Less ideal for:

- Small sample sizes (n < 100)
- Simple linear relationships (Cox PHM may suffice)
- Need for explicit feature effect estimates
- Limited computational resources
- Interpretability is paramount over prediction accuracy
- Quick prototyping (training can take hours/days)

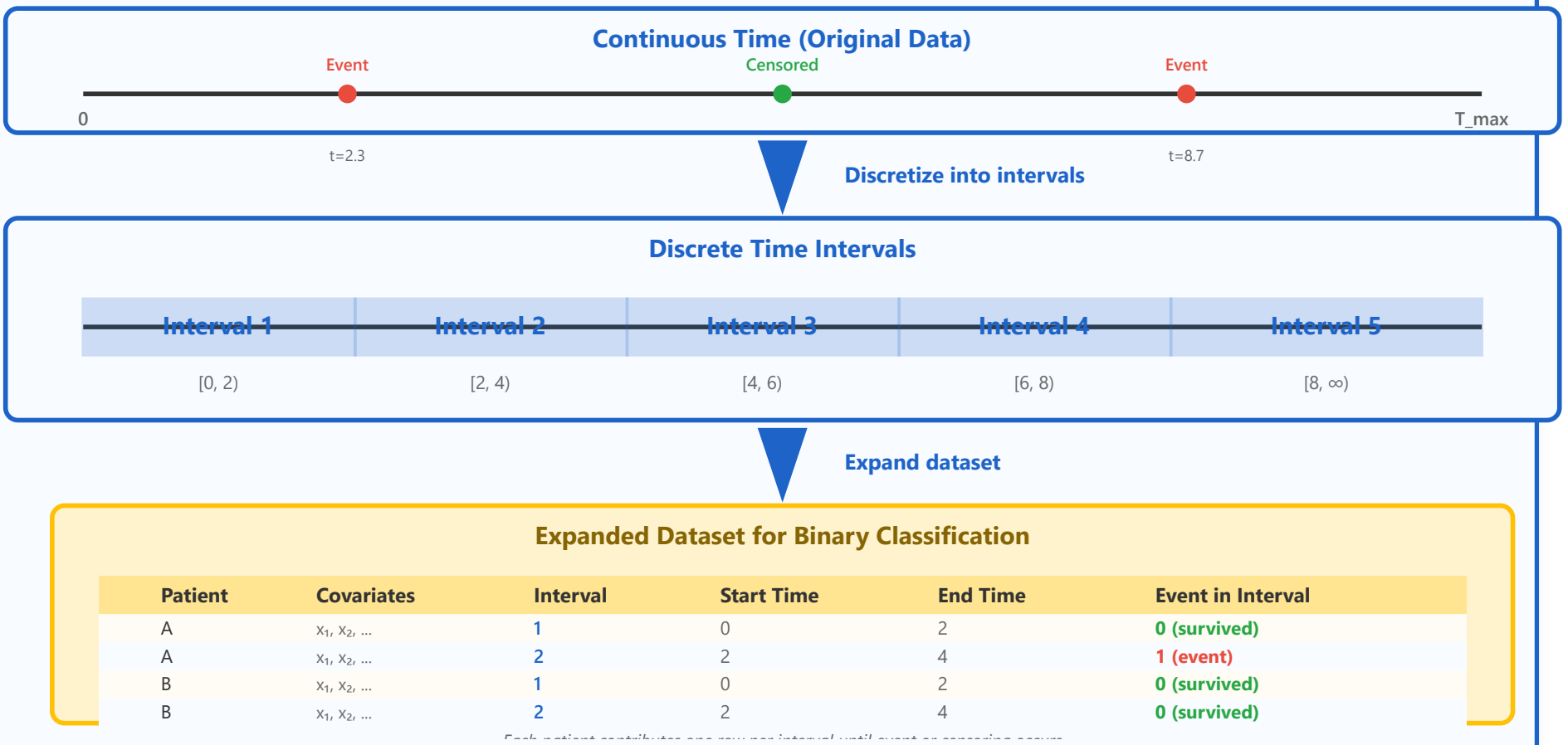
METHOD 3

Discrete Time Survival Models

Overview & Core Concept

Discrete time models reformulate survival analysis as a sequence of binary classification problems. Instead of modeling continuous time directly, the follow-up period is divided into discrete intervals, and for each interval, we predict whether an individual will experience the event during that specific interval, given they survived up to that point. This transforms the complex censored survival problem into a series of easier-to-model conditional probabilities: P(event in interval j | survived to interval j). The beauty of this approach is that it allows us to use any standard binary classification algorithm (logistic regression, random forests, gradient boosting, neural networks) while properly handling censored data through the data expansion process.

Discrete Time Framework: From Continuous to Intervals



Transformation from continuous survival data to discrete time format. Continuous timeline is divided into intervals, and the dataset is expanded so each patient contributes one observation per interval they are at risk. Binary outcome indicates whether event occurred in that interval.

▸ Mathematical Framework

The discrete time model estimates the **discrete hazard** $\lambda_j(X)$ for each interval j:

Discrete Hazard (probability of event in interval j):

$$\lambda_j(X) = P(T = t_j \mid T \geq t_j, X)$$

This is the conditional probability that an individual with covariates X experiences the event in interval j, given they survived until the start of that interval.

We model this using logistic regression:

$$\text{logit}[\lambda_j(X)] = \log[\lambda_j(X) / (1 - \lambda_j(X))] = \alpha_{0j} + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

where:

- α_{0j} = baseline log-odds for interval j (interval-specific intercept)
- β = covariate effects (shared across all intervals by default)
- Can allow time-varying effects: β_j different for each interval

Survival function at time t_j:

$$S(t_j|X) = P(T > t_j|X) = \prod_{i=1}^{j} [1 - \lambda_i(X)]$$

This is the product of surviving each interval from 1 to j.

Probability of event in interval j:

$$P(T = t_j|X) = \lambda_j(X) \times S(t_{j-1}|X) = \lambda_j(X) \times \prod_{i=1}^{j-1} [1 - \lambda_i(X)]$$

▸ Key Features & Advantages

🎯 Simplicity & Flexibility

Converts survival analysis into standard binary classification. Can use ANY classification algorithm: logistic regression, random forests, gradient boosting, neural networks, SVM. No specialized survival software needed.

⏱️ Time-varying Effects

Easily model non-proportional hazards by allowing covariate effects to vary across intervals. Include interval indicators or interval × covariate interactions.

📊 Interpretable Probabilities

Direct probability predictions for each interval. Easy to communicate: "30% chance of event in next 6 months given survival so far." More intuitive than hazard ratios for some audiences.

🔧 Easy Implementation

Simple data preparation (expand to person-period format). Standard ML libraries (scikit-learn, XGBoost) work directly. Mature ecosystem of tools and techniques.

📈 Handles Competing Risks

Naturally extends to competing risks by using multinomial classification instead of binary. Can model multiple event types simultaneously with separate probabilities.

⚡ Computational Efficiency

Training is typically faster than RSF or DeepSurv. Standard optimized libraries. Predictions are very fast: just evaluate classifier for each interval.

▸ Implementation Steps

Step-by-step guide to implementing discrete time models:

- Step 1 - Define Intervals: Choose interval boundaries based on domain knowledge or data distribution. Options: equal width (e.g., every 6 months), equal frequency (quantiles of event times), or clinically meaningful periods.

- **Step 2 - Expand Dataset:** Transform data to person-period format. Each individual contributes one row per interval they are at risk. Include covariates and interval identifier for each row.
- **Step 3 - Create Binary Outcome:** For each row, outcome = 1 if event occurred in that interval, 0 if individual survived the interval. Exclude rows after event or censoring.
- **Step 4 - Add Interval Indicators:** Include dummy variables or numeric indicators for each interval. This allows baseline hazard to vary across time. Can use one-hot encoding or ordinal encoding.
- **Step 5 - Fit Classification Model:** Train any binary classifier on expanded dataset. Logistic regression for interpretability, tree-based methods for flexibility, neural networks for complex patterns.
- **Step 6 - Predict Survival Curves:** For new individual, predict probability for each interval sequentially. Multiply survival probabilities: $S(t_j) = \prod_{i=1}^{j} (1 - \hat{P}_i)$ where $\hat{P}_i$ is predicted probability for interval i.
- **Step 7 - Model Evaluation:** Use time-dependent C-index, Brier score, or calibration plots. Can evaluate within specific intervals or overall. Assess calibration by comparing predicted vs. observed event rates.

⚙️ Design Choices & Considerations

Interval Selection: Too few intervals = loss of temporal resolution. Too many intervals = sparse data per interval, increased variance. Typical: 5-20 intervals. Use event distribution to guide choice.

Handling Ties: Multiple events at same time are naturally handled since they fall in same interval. Unlike Cox models which require special handling.

Time-varying Covariates: Easy to incorporate! Just include time-updated covariate values in each row. Example: treatment changes, lab values, biomarker levels.

Class Imbalance: Typically few events per interval = imbalanced classes. Use: class weights, SMOTE, focal loss, or stratified sampling to address.

Interval Effects: Shared effects (β constant) assumes proportional odds. Time-varying effects (β_j) allows non-proportional hazards. Trade-off: flexibility vs. overfitting.

▸ Practical Example & Code

```
# Python implementation with pandas and scikit-learn
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from lifelines.utils import concordance_index

def create_person_period(df, time_col, event_col, interval_breaks):
    """
    Expand survival data to person-period format

    Parameters:
    -----------
    df : DataFrame with survival data
    time_col : name of time column
    event_col : name of event indicator column (1=event, 0=censored)
    interval_breaks : list of interval boundaries [0, 2, 4, 6, 8, inf]

    Returns:
    --------
    Expanded DataFrame in person-period format
    """
    expanded_rows = []
```

```
    for idx, row in df.iterrows():
        time = row[time_col]
        event = row[event_col]

        # Determine which interval the event/censoring falls in
        interval_idx = np.searchsorted(interval_breaks, time, side='right') - 1

        # Create row for each interval up to event/censoring
        for j in range(interval_idx + 1):
            new_row = row.copy()
            new_row['interval'] = j
            new_row['interval_start'] = interval_breaks[j]
            new_row['interval_end'] = interval_breaks[j+1]

            # Event indicator for this interval
            if j < interval_idx:
                new_row['event_in_interval'] = 0  # Survived this interval
            else:
                new_row['event_in_interval'] = event  # Event or censored

            expanded_rows.append(new_row)

    return pd.DataFrame(expanded_rows)

# Example usage
# Original data
survival_data = pd.DataFrame({
    'time': [2.3, 5.1, 8.7, 3.2, 6.8],
    'event': [1, 0, 1, 1, 0],
    'age': [65, 72, 58, 61, 69],
    'treatment': ['A', 'B', 'A', 'B', 'A'],
    'biomarker': [2.3, 1.8, 3.1, 2.5, 1.9]
})
```

```
# Define intervals
interval_breaks = [0, 2, 4, 6, 8, np.inf]

# Expand to person-period format
pp_data = create_person_period(
    survival_data, 'time', 'event', interval_breaks
)

# Prepare features
# One-hot encode categorical variables
pp_data = pd.get_dummies(pp_data, columns=['treatment'])

# Create interval dummy variables (baseline hazard)
pp_data = pd.get_dummies(pp_data, columns=['interval'], prefix='int')

# Select features for modeling
feature_cols = ['age', 'biomarker', 'treatment_A', 'treatment_B',
                'int_0', 'int_1', 'int_2', 'int_3', 'int_4']
X = pp_data[feature_cols]
y = pp_data['event_in_interval']

# Fit logistic regression (or any classifier)
model = LogisticRegression(class_weight='balanced', max_iter=1000)
# Alternative: Gradient Boosting for non-linear relationships
# model = GradientBoostingClassifier(n_estimators=100)

model.fit(X, y)

# Predict survival curve for new patient
def predict_survival_curve(model, patient_data, interval_breaks):
    """Predict survival probability for each interval"""
    n_intervals = len(interval_breaks) - 1
```

```
    survival_probs = []
    survival_cum = 1.0

    for j in range(n_intervals):
        # Create features for interval j
        patient_data['interval'] = j
        patient_data_encoded = pd.get_dummies(
            patient_data, columns=['interval'], prefix='int'
        )

        # Predict probability of event in interval j
        prob_event = model.predict_proba(
            patient_data_encoded[feature_cols]
        )[0, 1]

        # Update cumulative survival
        survival_cum *= (1 - prob_event)
        survival_probs.append(survival_cum)

    return np.array(survival_probs)

# Predict for new patient
new_patient = pd.DataFrame({
    'age': [60],
    'biomarker': [2.0],
    'treatment': ['A']
})

surv_curve = predict_survival_curve(
    model, new_patient, interval_breaks
)
```

```
print("Survival probabilities by interval:", surv_curve)
```

📚 When to Use Discrete Time Models

Ideal scenarios:

- Time naturally discrete (monthly follow-ups, yearly events)
- Want to use standard ML libraries and tools
- Need time-varying covariate effects
- Prefer probability interpretations over hazards
- Have competing risks to model
- Want fast training and simple implementation
- Team familiar with classification but not survival analysis

Less ideal for:

- Precisely measured continuous time is critical
- Need smooth hazard function estimates
- Very fine time resolution required
- Small sample size with many intervals (overfitting risk)
- Strong preference for proportional hazards framework
- Want traditional hazard ratio interpretations

SECTION 5

Method Comparison & Selection Guide

Comprehensive Comparison Table

Aspect	Random Survival Forests	DeepSurv	Discrete Time Models
Core Approach	Ensemble of survival trees with bootstrap aggregation	Deep neural network with Cox loss	Sequential binary classification per interval
Assumptions	None (fully non-parametric)	Proportional hazards (like Cox)	Proportional odds (can relax with interactions)
Handles Non-linearity	✅ Excellent (tree splits)	✅ Excellent (deep layers)	✅ Good (depends on classifier choice)
Feature Interactions	✅ Automatic (implicit in trees)	✅ Automatic (hidden layers)	⚠️ Manual specification often needed
High-dimensional Data (p >> n)	✅ Good (random feature selection)	✅ Excellent (designed for this)	⚠️ Moderate (needs regularization)

Aspect	Random Survival Forests	DeepSurv	Discrete Time Models
Sample Size Required	Moderate (n > 100 preferable)	Large (n > 500 ideal)	Small-Moderate (n > 50 often sufficient)
Missing Data Handling	✅ Native support (surrogate splits)	❌ Requires imputation	❌ Requires imputation
Interpretability	⚠️ Variable importance, partial dependence	❌ Black box (can use SHAP/LIME)	✅ Good (especially with logistic regression)
Training Time	Moderate (parallelizable)	Long (GPU helps significantly)	Fast (standard classification)
Prediction Speed	Fast	Very fast (once trained)	Very fast
Implementation Complexity	Low (good packages available)	Moderate-High (need DL framework)	Low (standard ML tools)
Time-varying Effects	⚠️ Possible but complex	⚠️ Requires architecture changes	✅ Easy (interval × covariate interactions)
Best For	Moderate data, need variable importance, non-linear patterns	Large datasets, genomics, complex non-linearity, deep patterns	Discrete time, simple implementation, time-varying effects

▸ Decision Flowchart

When choosing a method, consider these key questions:

- **Q1: What is your sample size?**
  - Small (n < 100): Discrete Time Models or traditional Cox
  - Moderate (100 < n < 500): Random Survival Forests
  - Large (n > 500): DeepSurv becomes viable
- **Q2: How many features do you have relative to sample size?**
  - p >> n (high-dimensional): DeepSurv or RSF with feature selection
  - p < n (low-dimensional): Any method works
- **Q3: Do you suspect strong non-linear relationships?**
  - Yes: RSF or DeepSurv
  - No/Uncertain: Start with Discrete Time + Logistic Regression
- **Q4: How important is interpretability?**
  - Critical: Discrete Time with Logistic Regression
  - Moderate: RSF (variable importance available)
  - Not critical: DeepSurv (focus on prediction)
- **Q5: Do you need to model time-varying effects?**
  - Yes: Discrete Time Models (easiest)
  - No: Any method works
- **Q6: What computational resources do you have?**

- Limited: Discrete Time Models (fastest)
- Moderate: RSF (parallelizable)
- Substantial (GPU access): DeepSurv

💡 Practical Recommendations

For most applications, start with Random Survival Forests: Good balance of performance, interpretability, and ease of use. Handles non-linearity well and provides variable importance.

Use DeepSurv when you have: Very large sample size (n > 1000), high-dimensional features (genomics, images), or complex non-linear patterns that RSF doesn't capture well.

Choose Discrete Time Models when: Time is naturally discrete, you need simple implementation with standard tools, or you want to model time-varying covariate effects easily.

Best practice: Try multiple methods! Use cross-validation to compare C-index, calibration, and Brier scores. The "best" method is dataset-specific. Ensemble predictions from multiple methods often works best.

🎯 Key Takeaways

- **Random Survival Forests** offer robust, non-parametric predictions with automatic variable importance. Great default choice for moderate-sized datasets.
- **DeepSurv** leverages deep learning to capture highly complex patterns in high-dimensional data. Best for large samples and genomic/imaging applications.
- **Discrete Time Models** simplify survival analysis to binary classification, enabling use of any ML algorithm. Ideal for discrete time and time-varying effects.
- **No universally "best" method** exists. Choice depends on: sample size, dimensionality, interpretability needs, computational resources, and domain requirements.
- **Evaluation is critical:** Use time-dependent C-index, calibration plots, and Brier scores. Validate on held-out data or via cross-validation.
- **Consider ensembles:** Combining predictions from multiple methods often improves performance and robustness.

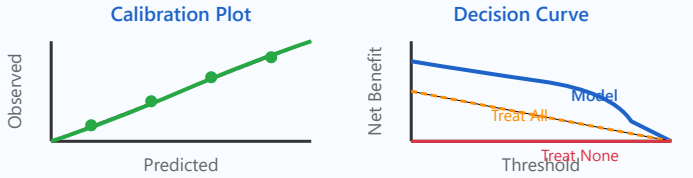
Clinical Risk Scores

Translate complex models into simple, actionable scoring systems

Development Pipeline

1	Select Predictors	Clinical relevance + Statistics
2	Fit Regression	Logistic/Cox regression
3	Convert to Points	β → Integer scores
4	Create Tool	Nomogram/Calculator
5	Validate Externally	Different population

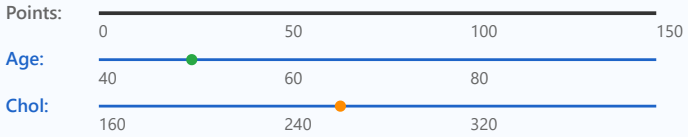
Validation Metrics



Performance Requirements:

- C-index > 0.7
- Good calibration
- Net benefit +
- External val

Example Nomogram: Cardiovascular Risk



Total: 75 points
→ 15% 10-year risk

Low | Med | High

Risk Categories

✓ Clinical Examples

APACHE II (ICU mortality) • Framingham Risk Score (CVD) • MELD Score (liver transplant) • GRACE Score (ACS) • CHA_2DS_2-VASc (stroke risk)

📊 Mathematical Principles Behind Risk Scores

β Coefficient → Point Conversion

Concrete Example: CVD Risk

Step 1: Logistic Regression Model

$\log(p/(1-p)) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots$

Step 2: Scale to Integer Points

$\text{Points}_i = \text{round}(\beta_i / \beta_{min} \times \text{constant})$

Step 3: Total Score = Σ Points → Probability

Example: Age coefficient $\beta = 0.05$
→ Points per year = $0.05/0.01 \times 2 = 10$

Variable	β Coefficient	Points
Age (per 10y)	$\beta = 0.50$	+25 pts
Smoking (Yes)	$\beta = 0.80$	+40 pts
Diabetes (Yes)	$\beta = 0.60$	+30 pts
Hypertension	$\beta = 0.40$	+20 pts

Patient: Age 60, Smoker, Diabetic

Total = 50 + 40 + 30 = 120 points

→ 10-year CVD risk = 28%

Score → Probability Conversion Methods

Method 1: Lookup Table

Total Score	Risk %
0-40	<5%
41-80	5-15%
81-120	15-30%
>120	>30%

✓ *Simple, clinician-friendly*

Method 2: Direct Formula

$p = 1 / (1 + e^{-LP})$

LP = linear predictor from score

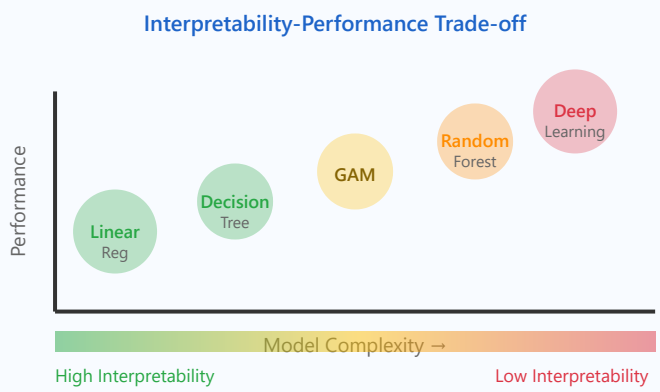
Score = 120 → LP = 0.06×120 - 3.5
→ p = 28.3%

Key Insight: Risk scores transform complex regression models into simple integer addition. The scaling factor is chosen to balance precision (enough granularity) with simplicity (easy mental math). Typically, total scores range 0-200 points for optimal usability.

Interpretable ML for Clinical Adoption

⚠ **Black Box Problem:** Clinicians won't trust models they can't understand

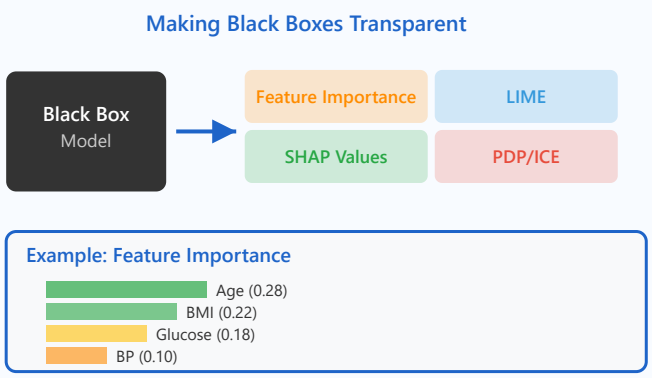
Glass Box Models



- Linear/Logistic Regression
- Decision Trees
- GAMs (Generalized Additive Models)
- Rule-based systems

Inherently interpretable

Post-hoc Explanations



- Feature Importance (RF, XGBoost)
- Partial Dependence Plots
- LIME (Local Interpretable)
- SHAP Values

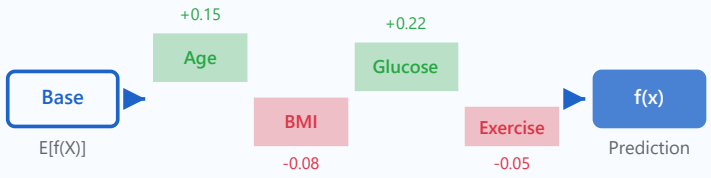
Explain black boxes

Clinical Acceptance Requires: Model explanation + Clinical validation + Physician trust + Regulatory approval

SHAP Values for Model Interpretation

SHapley Additive exPlanations - unified framework for interpretability

How SHAP Works

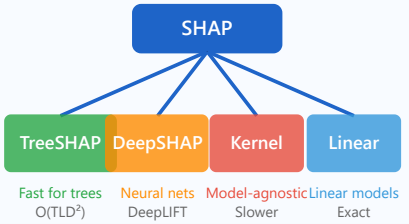


$$f(x) = \varphi_0 + \varphi_1 + \varphi_2 + \ldots + \varphi_n$$

Shapley Value: Fair contribution from cooperative game theory

- Considers all possible feature combinations
- Satisfies consistency & local accuracy

SHAP Algorithms



Key Properties: ✓ Local accuracy ✓ Missingness ✓ Consistency

The only method satisfying all desired properties (Lundberg & Lee, 2017)

Visualization Types

Clinical Validation Framework

Internal Validation

Cross-validation
Bootstrap
Same institution data

External Validation

Different hospitals
Different populations
Geographic diversity

Prospective Studies

Real-time predictions
Clinical workflow
RCT if possible

⚠ Real-World Performance

Models often degrade when deployed - monitor performance continuously!

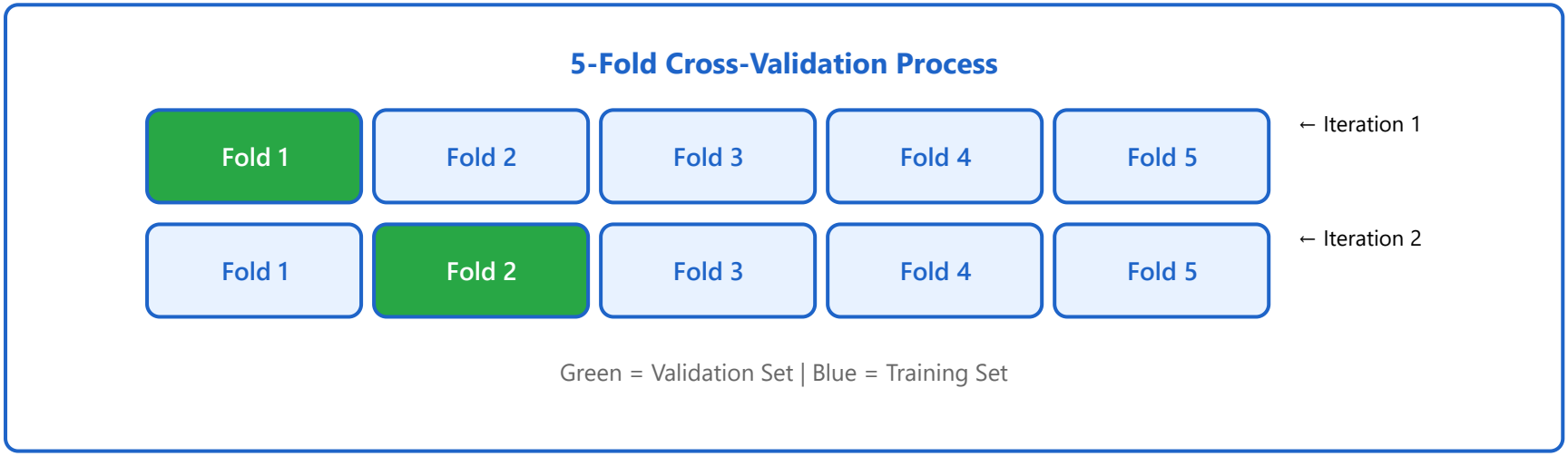
FDA Requirements for Medical AI: Multi-site validation, diverse populations, clinical outcomes

📊 Internal Validation: Building Confidence

1. K-Fold Cross-Validation

Divides the dataset into K equal parts (folds). The model is trained on K-1 folds and validated on the remaining fold. This process repeats K times, with each fold serving as the validation set once.

Example: 5-fold CV with 1,000 patients → 800 training, 200 validation per fold



2. Bootstrap Validation

Random sampling with replacement from the original dataset. Provides robust estimates of model performance and confidence intervals. Typically performed 500-1,000 times.

Example: From 1,000 samples, create 1,000 bootstrap samples, each with ~632 unique cases

🎯 Key Advantages:

- Maximizes use of limited data
- Provides confidence intervals for performance metrics

- Detects overfitting before external testing
- Cost-effective initial validation step

🌍 External Validation: The Real Test

1. Geographic External Validation

Testing the model on data from different hospitals, regions, or countries. This addresses variations in patient demographics, disease prevalence, and clinical practices.

Example: Model trained at US academic center → validated at European community hospital

Multi-Site Validation Scenario

Site A (Training)	Site B (External)	Site C (External)
• Urban academic center	• Community hospital	• International site
• 80% training data	• External validation	• Different population
• n = 5,000 patients	• n = 2,000 patients	• n = 1,500 patients
• AUC: 0.92	• AUC: 0.87 ✓	• AUC: 0.85 ✓

2. Temporal External Validation

Validating on data collected after the training period. This tests whether the model remains accurate as clinical practices, demographics, and disease patterns evolve over time.

Example: Train on 2015-2020 data → validate on 2021-2023 data

⚠ Common Challenges:

- **Domain shift:** Different patient populations, equipment, protocols
- **Data quality:** Inconsistent coding, missing values, measurement differences
- **Performance degradation:** Expect 5-15% decrease in accuracy

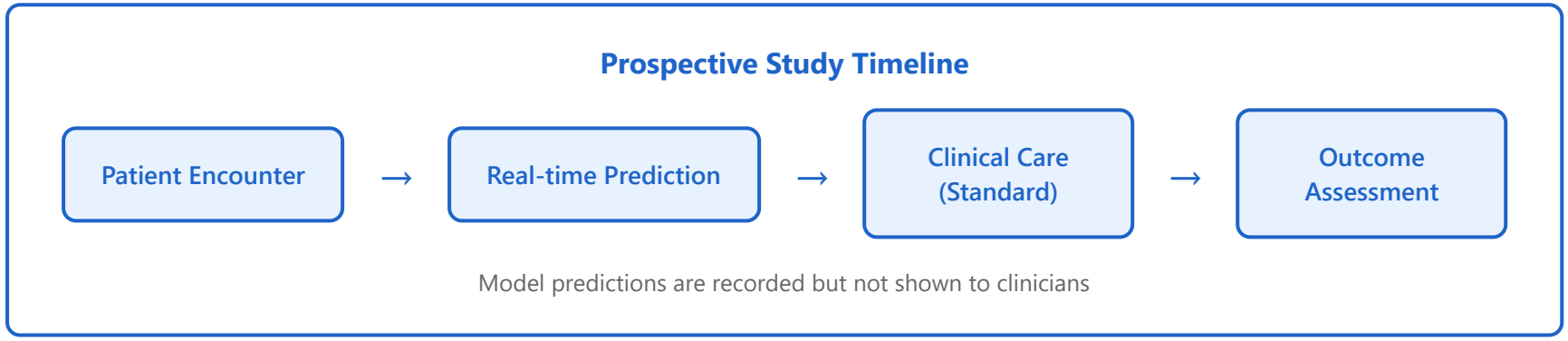
- **Calibration issues:** Predicted probabilities may not match actual outcomes

🚀 Prospective Studies: Clinical Reality

1. Prospective Observational Study

The model makes predictions in real-time as new patients are encountered, but clinicians are not required to act on the predictions. This evaluates real-world performance without influencing clinical decisions.

Example: Silent mode deployment for 6 months, comparing predictions to actual outcomes



2. Randomized Controlled Trial (RCT)

The gold standard for clinical validation. Patients are randomly assigned to receive AI-assisted care or standard care. This directly measures the impact of the AI system on clinical outcomes and decision-making.

Example: 1,000 patients → 500 with AI alerts, 500 standard care, compare mortality rates

Primary Outcomes
Mortality, morbidity, length of stay, readmissions

Secondary Outcomes
Diagnostic accuracy, treatment appropriateness, cost

Process Metrics
Time to diagnosis, alert compliance, workflow impact

Safety Metrics
False alarms, missed cases, adverse events

✅ Critical Success Factors:

- Integration with clinical workflow (minimal disruption)
- Clinician training and acceptance
- Continuous monitoring of model performance
- Clear protocols for alert fatigue management
- Regulatory compliance and ethical approval

📋 Validation Hierarchy & Requirements

Validation Type	Strength of Evidence	Time & Cost	Key Limitation
Internal	⭐ Low	Days-Weeks / $	Same data distribution
External	⭐⭐⭐ Moderate-High	Months / $$	Retrospective bias
Prospective	⭐⭐⭐⭐ High	6-24 Months / $$$	Time-intensive
RCT	⭐⭐⭐⭐⭐ Highest	1-3 Years / $$$$	Cost & complexity

Recommended Validation Pathway

1. Internal Validation → 2. External Validation → 3. Prospective Study → 4. RCT (if warranted)

Each step builds evidence for clinical adoption and regulatory approval

🔄 Post-Deployment Monitoring

Continuous validation is essential:

- Monitor performance metrics monthly
- Track data drift and model degradation
- Update models as clinical practices evolve

- Maintain audit trails for regulatory compliance

Hands-on: scikit-learn for Biomedical Data

Pipeline Creation

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([
  ('scaler', StandardScaler()),
  ('selector', SelectKBest()),
  ('clf', LogisticRegression())
])
```

Model Selection

```
from sklearn.model_selection import
  GridSearchCV
gs = GridSearchCV(pipe, params,
  cv=StratifiedKFold(5),
  scoring='roc_auc')
```

Practice Tasks

- Load biomedical dataset
- Handle missing values
- Scale features appropriately
- Deal with class imbalance
- Build classification pipeline
- Perform nested CV
- Generate evaluation report
- Plot ROC and PR curves

Hands-on: lifelines for Survival Analysis

Python library for survival analysis in clinical research

Kaplan-Meier

```
from lifelines import KaplanMeierFitter
kmf = KaplanMeierFitter()
kmf.fit(T, E, label='Group A')
kmf.plot_survival_function()
```

Cox Regression

```
from lifelines import CoxPHFitter
cph = CoxPHFitter()
cph.fit(df, 'T', 'E')
cph.print_summary()
```

Practice Tasks

- Load clinical trial data
- Plot KM curves by treatment
- Fit Cox model
- Test PH assumption
- Calculate C-index
- Make predictions

Thank You!

Questions? Let's discuss!

Next: Practice and case studies

Office Hours: By appointment