→ $\boxed{\text{Bagging}}$

- **What is Bagging**
  Bagging is an ensemble learning technique that trains multiple models independently on different bootstrap samples and combines their predictions to improve stability and accuracy.

- **Purpose (When)**
  To reduce variance and overfitting in high variance models (especially Decision Trees).

- **Why it is used**
  - Stabilizes unstable models
  - Reduces overfitting
  - Improves generalization
  - Parallelizable

- ~~Ho~~ **How it works (Algorithm)**
  - Create multiple bootstrap samples (sampling with replacement)
  - Train a base learner on each sample
  - Make predictions independently
  - Aggregate results: Classification → majority vote
    Regression → mean

- **Formula**
  - Classification : $\hat{y} = \text{mode}(h_1(x), h_2(x), \ldots, h_n(x))$
  - Regression : $\hat{y} = \frac{1}{n} \sum h_i(x)$

- Technical Details
  - Ensemble type: Parallel
  - Reduces variance, not bias
  - Base models are independent

- Parameters
  - Number of estimators
  - Base estimator type (gini / entropy)
  - Bootstrap size (depth)

- Pros
  - Reduces overfitting
  - Simple and effective
  - Parallel execution

- Cons
  - No bias reduction
  - Less interpretable
  - Higher computation

- Real World Application (Where)
  - Financial risk models
  - Medical prediction systems
  - Baseline ensemble models.

→ Gradient Boosting

- **What is Gradient Boosting**
  Gradient Boosting is a sequential ensemble technique where each new model corrects the errors of the previous models using gradient descent.

- Purpose ~~(Why)~~ (When)
  To reduce bias and build strong predictive models from weak learners.

- **Why it is used**
  - Handles complex non-linear patterns
  - High predictive accuracy
  - Flexible loss function

- **How it works (Algo)**
  - Start with a simple model (initial prediction)
  - Compute residual errors
  - Train next model on residuals
  - Add model to ensemble with learning rate
  - Repeat sequentially

- Formula
  $F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$ ∵ η = learning rate
  ∵ h_m = weak learner

- Technical Details
  - Ensemble type: Sequential
  - Optimizes arbitrary loss functions
  - Uses gradient descent in function space

- Parameters
  - Learning rate
  - Number of estimators
  - Tree depth
  - Loss function

- Pros
  - High accuracy
  - Handles complex data
  - Custom loss functions

- Cons
  - Slow training
  - Sensitive to hyperparameters
  - Prone to overfitting if not tuned.

- Real World Applications (where)
  - Fraud detection
  - Ranking systems
  - Credit scoring

→ $\boxed{\text{AdaBoost (Adaptive Boosting)}}$

- What is AdaBoost
~~Aad~~ AdaBoost is a boosting algorithm that focuses on misclassified samples by increasing their weights in ~~seqt~~ subsequent models.

- Purpose (When)
To convert weak learners into a strong classifier by focusing on hard examples.

- Why it is used
  - Simple boosting algorithm
  - Strong theoretical foundation
  - Works well with weak learners

- How it works (Algo)
  - Assign equal weigths to all samples
  - Train weak learner
  - Increase weights of misclassified points
  - Train next learner on reweighted data
  - Combine learners using weighted voting.

- Formula
  - Final model: $F(x) = \sum \alpha_m h_m(x)$
  - Learner weight: $\alpha_m = \frac{1}{2} \ln \frac{1 - error}{error}$

- Technical Details
  - Sequential learning
  - Sensitive to noisy data
  - Uses exponential loss

- Parameters
  - Number of estimators
  - Learning rate
  - Base estimator

- Pros
  - Improves weak models
  - Simple implementation
  - Less overfitting than trees

- Cons
  - Sensitive to noise & outliers
  - Poor performance on complex data
  - Not scalable

- Real World Applications (Where)
  - Face detection
  - Text classification
  - Early ML systems

DATE: / /
PAGE:

→ XGBoost (Extreme Gradient Boost)

- **What is XGBoost**
XGBoost is an optimized, scalable implementation of Gradient Boosting designed for speed, performance and regularization.

- **Purpose (When)**
To achieve state of the art accuracy on structured/tabular data.

- **Why it is used**
  - Extremely fast
  - Built in regularization
  - Handles missing values
  - Winner of many ML competitions

- **How it works (Algo)**
  - Builds trees sequentially
  - Uses second-order derivatives (Hessian)
  - Applies regularization to trees
  - Prunes trees automatically
  - Parallelizes tree construction

- **Formula**
Objective Function: $Obj = \sum l(y_i, \hat{y}_i) + \sum \Omega(f_k)$
Regularization: $\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum w^2$

DATE: \ \
PAGE:

- Technical Details
  - Uses newton boosting
  - Supports sparse data
  - Cache-aware and parallel

- Parameters
  - nestimators : no of trees
  - learning_rate : step size
  - max_depth : tree depth
  - subsample : row sampling
  - colsample_bytree : Feature sampling
  - lamda, alpha : regularization

- Pros
  - Very high accuracy
  - Built in regularization
  - Handles missing values
  - Scalable

- Cons
  - Complex tuning
  - Less interpretable
  - Overkill for small datasets

- Real World Applications (Where)
  - Kaggle competitions
  - Finance risk models
  - Search ranking
  - Recommendation systems