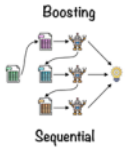
# Sequential Ensemble

08 December 2024 18:15

↳ Generates base learners in a sequence
- ↳ Each model corrects errors of prev. model
  - ↳ Higher weights to learners that were previously misrepresented.
- imbalanced datasets. { → reduce bias, → improve accuracy
- Train models sequentially. = BOOSTING

But how does boosting work?
- → fit an initial model.
- → second model focuses on accurately predicting cases where model 1 performed poorly.
- → Combination of these models is better than individual model
- → Repeat

## Weights + Training Instances :
- Weight of an instance = importance of classifying correctly.
- weights are adjusted w.r.t each iteration

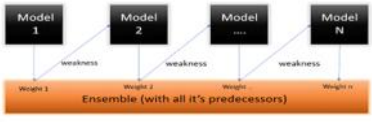

| | Bagging | Boosting |
|---|---|---|
| Similarities | • Uses voting<br>• Combines models of the same type | |
| Differences | Individual models are built separately | Each new model is influenced by the performance of those built previously |
| | Equal weight is given to all models | Weights a model's contribution by its performance |

w.r.t misclassified instances — more weight
correctly classified — less weight.
(normalize weights to sum to 1)

Disadvantages :
- vulnerable to outliers
- Difficult for real time
- Computationally expensive.

## Adaboost

- Stagewise addition — multiple weak learners → strong learners.
  - → decision stumps (single level dt)
- $\alpha$ indirectly prop. to error of weak learner




- Working mechanism :
  - → Observer weight initialized equally
  - → Misclassified samples = increased weights.
  - → Repeat until reach desired no. of iterations.

} Final = weighted sum of weak learners.
weight ↔ accuracy.

## Gradient Boosting

- Stagewise addition
- Working mechanism :
  - → First learner — no dataset - mean of relevant column

- Working mechanism:
  - → First learner — no dataset — mean of relevant column
  - → Calculate residual, use as target column for next weak learning algo.
  - → Keep going. Just ensure numerical/categorical data.

# Newton Boosting

- Progressively reduce error by learning from residuals like GB.
  BUT.
  GB uses first derivative of loss fn for updates.
  NB calculates second derivative → Newton Rhapson ← gradient = 1st deriv, curvature = 2nd deriv } direction + rate of change

- Working mechanism
  - → initialize w/ decision tree to make initial preds.
  - → calculating residuals using derivatives.
    - residual = diff b/w actual & pred values
    - gradient for direction + hessian for curvature.
  - → Newton Rhapson update system
    - update = − gradient / hessian } faster convergence
    But additional calculations for second derivative increase load.

# XG Boost

- Extreme gradient boosting → key feature = efficient handling of missing values + parallel processing.

Key Innovations:

- Regularization: Penalizes model complexity to avoid overfitting, unlike traditional Gradient Boosting.
- Tree Pruning: Uses a "depth-wise" tree growth to prevent overfitting and enhance performance.
- Parallel Processing: Takes advantage of parallel computation, making it significantly faster than other boosting methods.
- Handling Missing Values: Has a built-in approach to handle missing data, assigning default directions based on best-fit.

- Working mechanism
  - → starts w/ initial weak learner (simple tree)
  - → Each tree fit to residuals of prev. tree. (take Newton Boost approach)
  - → Weighted tree splits w/ regularization
    (max. info gain; no overfitting)
  - → Pruning: for trees that split w/o accuracy increase.
  - → Combine trees based on weighted vote.

- Advantages
  - → speed + scalability
  - → performs well w/ structured data
  - → Handles imbalanced data.

Some key hyperparameters in XGBoost:

- Learning Rate (eta): Controls the size of steps during optimization. Lower values may require more trees but can lead to better performance.
- Max Depth: Limits the maximum depth of trees, balancing between capturing complex patterns and avoiding overfitting.
- Min Child Weight: Controls the minimum sum of instance weight in a child; higher values make the model more conservative.
- Gamma: Minimum loss reduction required for a split, which helps prevent unnecessary splits.
- Subsample: Controls the fraction of samples used per tree; lower values add randomness, helping to avoid overfitting.