Performance measures using Bagging

The remaining one-third of unused observations are called **out-of-bag** (OOB) observations.

OOB Error: Estimates test error using unused observations (1/3 per tree).

- Tree-based methods provide a natural metric of feature importance.
- **Mechanisms:**
  - *Impurity Reduction:* Assessment based on metrics such as Gini impurity or entropy decrease.
  - *Weighted Contributions:* Importance is weighted by the number of samples reaching each node.

Ensemble Methods: Random Forests

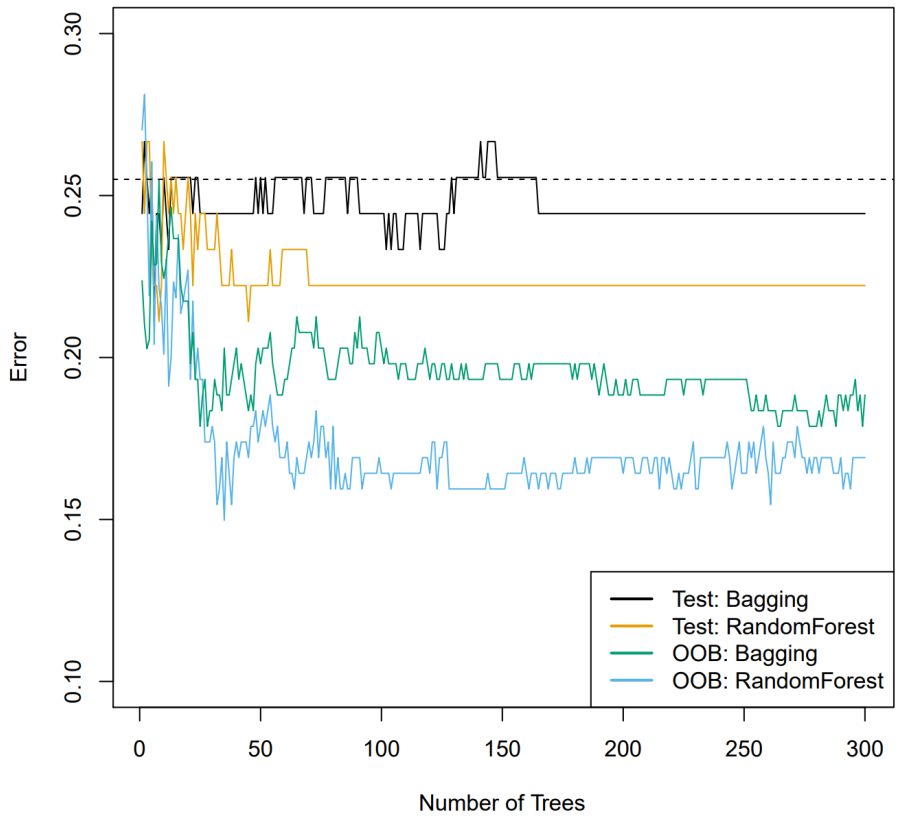
- Random forests improve bagged trees by **decorrelating** the trees.

Ensemble Methods: Random Forests

- Random forests improve bagged trees by **decorrelating** the trees.
- Averaging uncorrelated quantities reduces variance more effectively than averaging correlated ones.
- Random forests achieve this by considering only a subset of predictors for each split.
- At each split, a **random sample** of m predictors is selected as split candidates from the full set of p predictors.
- Typically, $m \approx \sqrt{p}$ is chosen, ensuring most predictors are excluded at each split.

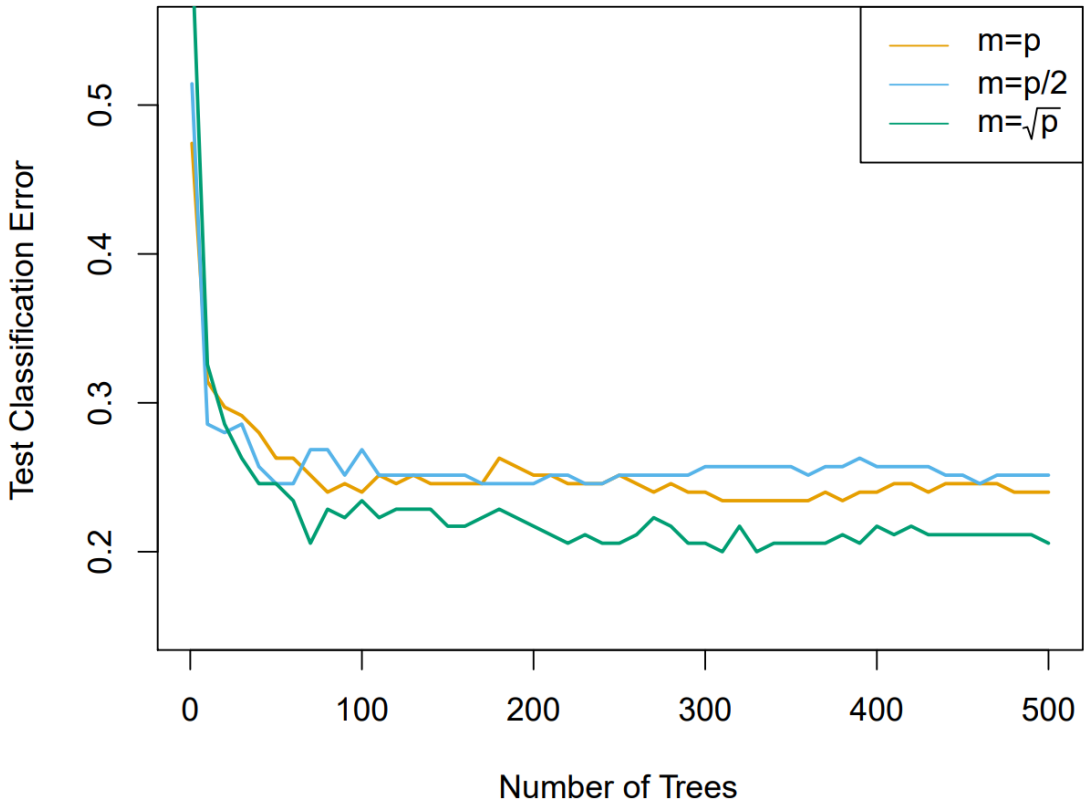
Ensemble Methods: Random Forests

- Bagging and random forests differ in the size of the predictor subset, m.
- Setting $m = p$ in random forests is equivalent to bagging.
- Choosing $m \approx \sqrt{p}$ in random forests reduces both test error and OOB error compared to bagging.



Ensemble Methods: Random Forests

- A small m in random forests is useful for datasets with many correlated predictors.
- Example: The effect of m on an example dataset divided into training and test sets.



Random Forests & Extra-Trees: An Overview

Random Forests

- Ensemble of decision trees trained on bootstrapped samples.
- Optimal split selection at each node.
- Tends to lower bias with a potential risk of higher variance.

Extra-Trees (Extremely Randomized Trees)

- Splitting nodes using randomized thresholds.
- Offers faster training, making it suitable for larger datasets.
- Trades off some accuracy (potentially higher bias) for speed.

- **Key Differences:**
  - Uses random thresholds to decide splits.
  - Often faster training compared to traditional Random Forests due to less rigorous split searching.

Boosting: Sequential Learning

- **Key Idea**: Grow trees sequentially to correct residuals. Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- **Algorithm (Regression)**:
  1. Initialize $\hat{f}(x) = 0$, residuals $r_i = y_i$.
  2. For $b = 1$ to B:
     - Fit tree $\hat{f}^b$ with d splits to (X, r).
     - Update model: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.
     - Update residuals: $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$.

Shrinkage Parameter λ

Slows learning to avoid overfitting (e.g., $\lambda = 0.01$).

Note: Each new tree attempts to capture signal that is not yet accounted for by the current set of trees.

Boosting parameters

- The number of trees, B: Boosting can overfit if B is too large, though overfitting occurs slowly. Use cross-validation to select B.
- The shrinkage parameter, λ: A small positive value controls the learning rate. Typical values are 0.01 or 0.001. Smaller λ may require larger B for good performance.
- The number of splits, d: Controls the complexity of the model. Often $d = 1$ works well, fitting an additive model where each term involves a single variable. More generally, d is the **interaction depth** of boosted models, involving at most d variables.

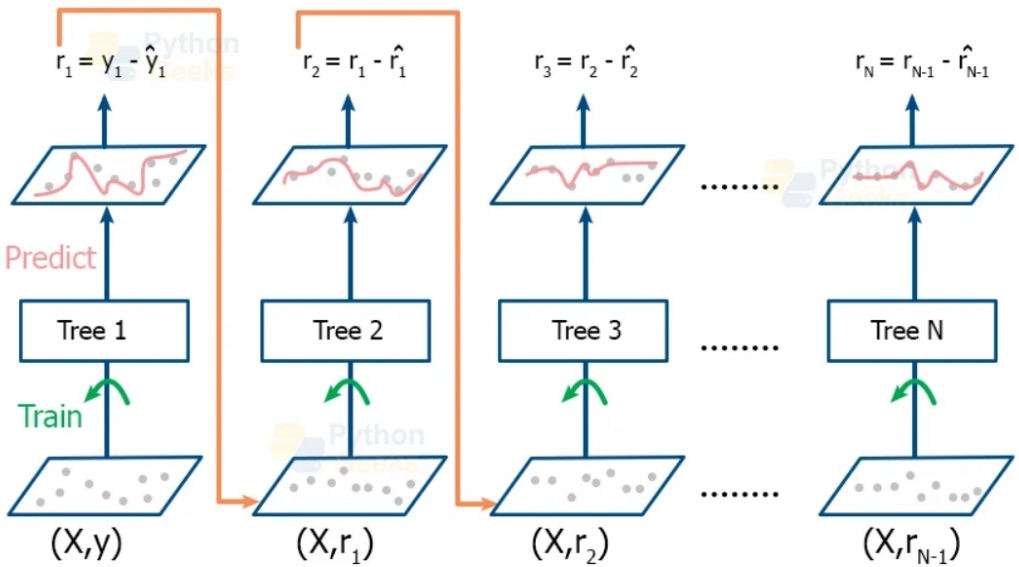
Limitations:

- Risk of overfitting if B is too large.
- Sequential training $\Rightarrow$ computationally intensive.

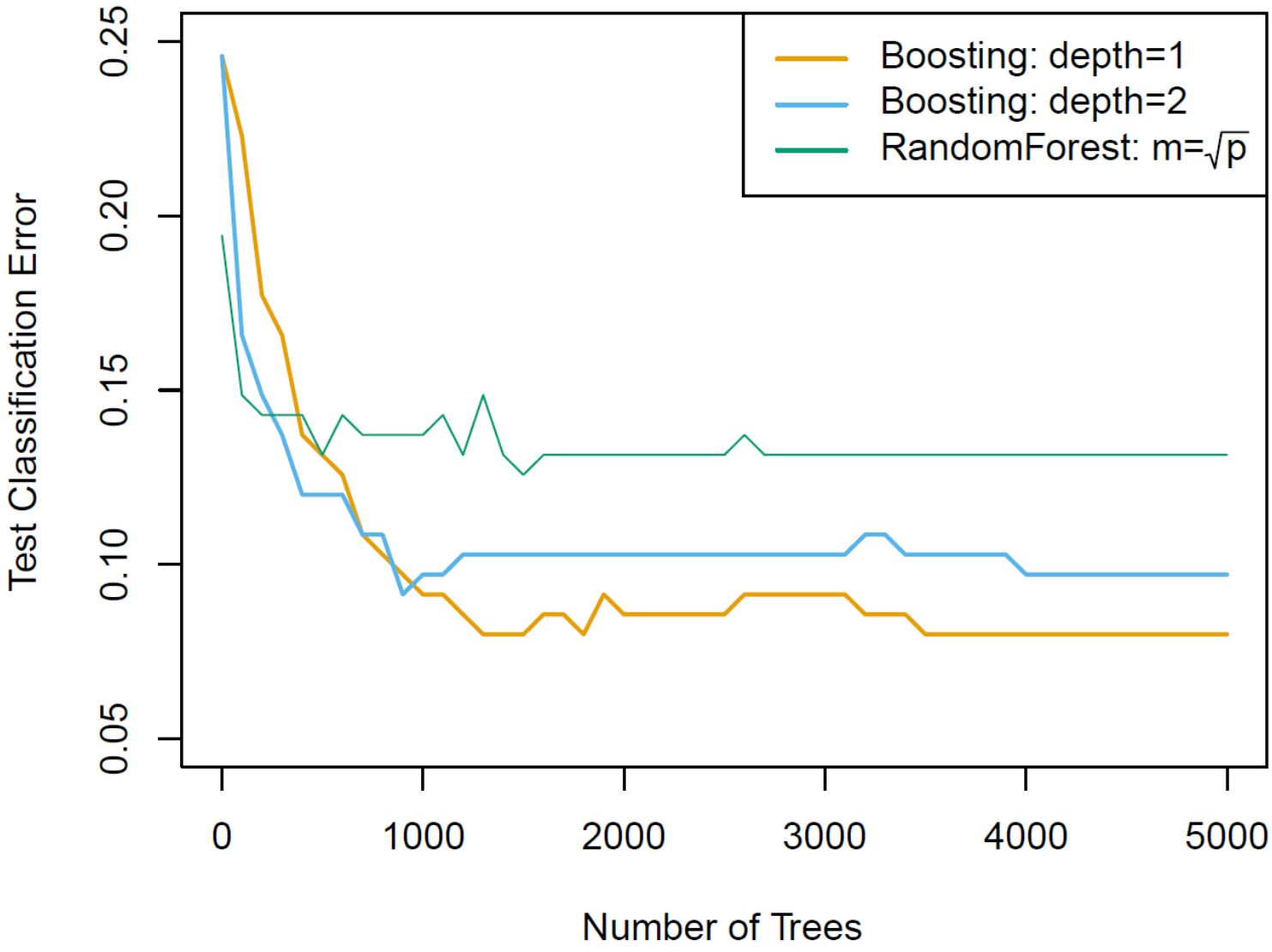
Gradient Boosting: Iterative Improvement

- **Idea:** Fit new models to the residuals (errors) produced by previous models.
- **Process for Regression Tasks:**
  1. Train an initial tree on the target variable y.
  2. Compute residuals: $y_2 = y - \text{tree}_1.\text{predict}(X)$.
  3. Train additional trees on these residuals.
  4. Final prediction is the sum of the prediction from all trees.

Working of Gradient Boosting Algorithm



Boosting Example result



AdaBoost: Adaptive Boosting

- **Concept:** Sequentially train weak learners; focus on examples misclassified by previous models.
- **Key Mechanisms:**
  1. Increase weights on misclassified instances.
  2. Use a weighted vote for final predictions.
- **Core Equations:**
  - Weighted Error rate:

$$r_j = \sum_{i=1}^{m} w(i) \text{ for i: } \hat{y_j}(i) \neq y(i)$$

  - Predictor weight:

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

  - Instance weight update for misclassified samples:

$$w^{(i)} \leftarrow w^{(i)} \exp(\alpha_j)$$

Optimized Boosting: HistGradientBoosting

- **Motivation:** Efficiently handle very large datasets.
- **Mechanism:**
  - Data is binned into histograms which speeds up split finding.
  - Complexity improves to approximately $O(b \times m)$, where b is the number of bins.
- **Advantages:**
  - Faster model training.
  - Better handling of categorical variables and missing data.

BART: Bayesian Framework

- **Key Idea**: Combine Bayesian inference with tree ensembles.
- **Mechanism**:
  - Initialize K trees to mean response.
  - Perturb trees iteratively via MCMC to fit partial residuals.
- **Prediction**: Average post burn-in (L iterations):

$$\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$$

Partial Residual

For k-th tree in iteration b:

$$r_i = y_i - \sum_{k' \neq k} \hat{f}_{k'}(x_i)$$

BART: Results & Robustness

Heart Data:

- Test/train error stabilizes after burn-in ($L = 100$).
- Minimal overfitting (small train-test error gap).

Advantages:

- Quantifies uncertainty via posterior intervals.
- Strong out-of-box performance (e.g., $K = 200$, $B = 1000$).

Comparison with Boosting:

- BART avoids overfitting seen in boosting (test error rises post-400 trees).

BART Example result

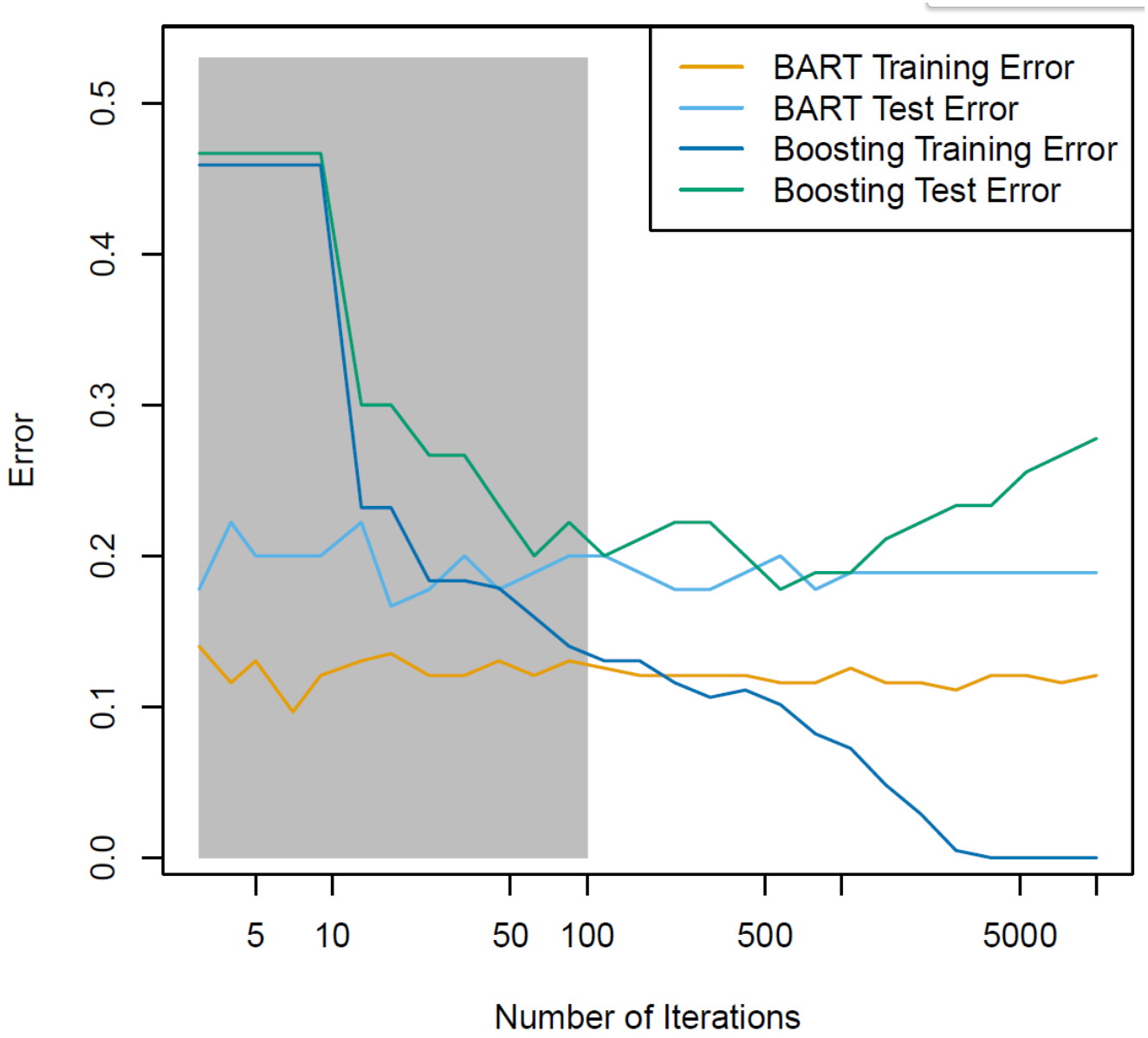


Table of Contents

1. Support Vector Machines
2. Tree-based Methods
3. Ensemble Learning
   - BART (Bayesian Additive Regression Trees)
4. Summary

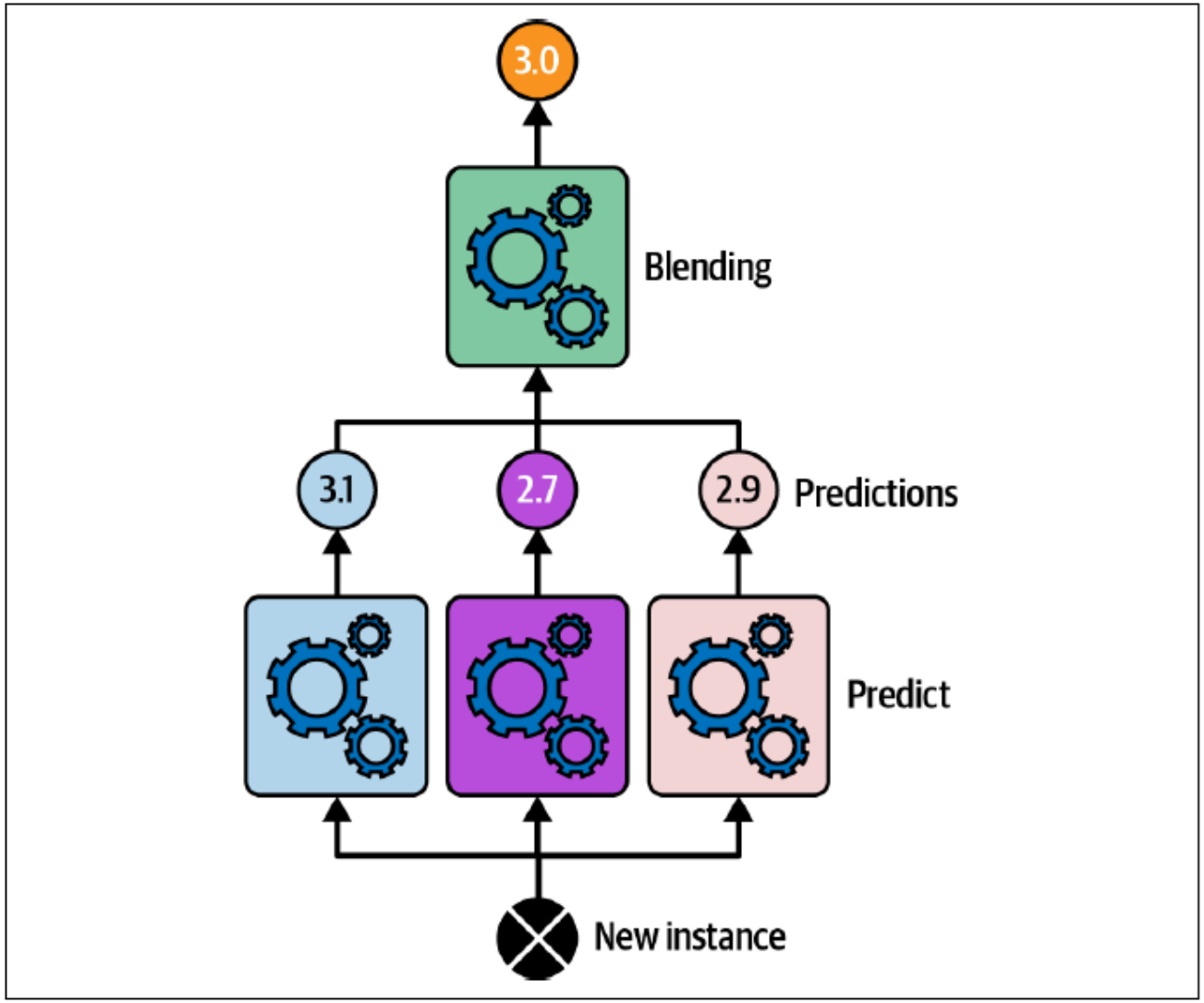
Summary

Method	**Key Feature**
Bagging	Bootstrap averaging, OOB error computation
Random Forests	Decorrelated trees via $m \approx \sqrt{p}$ splits
Boosting	Sequential residual correction, λ shrinkage
BART	Bayesian MCMC, tree perturbations, burn-in

Stacked Generalization: Combining Multiple Learners

- **Overview:** Use a *meta-learner* to integrate predictions from several base models.
- **Training Process:**
  1. Generate out-of-sample predictions via cross-validation.
  2. Train a meta-model (or blender) on these predictions.
  3. Optionally, retrain base models on the full dataset for the final ensemble.
- **Benefits:** Captures complex relationships not detected by individual models.

Stacked Generalization: Combining Multiple Learners



Practical Considerations and Best Practices

- **Validation:** Always use cross-validation to gauge model performance.
- **Hyperparameter Tuning:** Balance bias, variance, and computational cost via grid or random search.
- **Interpretability:** Take advantage of feature importance metrics in tree-based models for model insights.
- **Real-World Focus:** Consider the underlying data characteristics when choosing an ensemble method.