Binary Classification: Is Boosting stronger than Bagging?

DISCLAIMER: Summarized by AI

Problem they are trying to solve / Purpose of method

The paper aims to challenge the common assumption that boosting methods (e.g., XGBoost) are inherently superior to bagging methods (e.g., Random Forests) in binary classification tasks.

Previous problems that need to be solved:

- Vanilla Random Forests assume equal importance for all samples and trees.
- They lack mechanisms to focus on hard examples during training.
- Both boosting and bagging models suffer from poor interpretability.
- Boosting typically outperforms bagging in predictive accuracy, but at the cost of explainability and sensitivity to hyperparameters.

Why is the method introduced/needed?

- To enhance Random Forests with better performance and partial interpretability.
- To introduce **sample weighting** (favoring hard examples) and **model weighting** (personalized tree weights per prediction).
- To make bagging competitive with boosting in terms of performance, while being more interpretable and less reliant on tuning.

How does it differ from other methods?

What makes this method unique?

- Introduces **Enhanced Random Forests (ERF)**, which:
  - Use adaptive **sample weighting** to focus training on harder examples.
  - Use **model weighting**, where only the most relevant trees (based on nearest-neighbor similarity) are used for each prediction.
  - Enable **partial interpretability** by identifying the small set of trees that most influence a prediction.
- Unlike boosting (e.g., XGBoost), ERF does not build on residuals and maintains intuitive decision logic.
- Outperforms both vanilla Random Forests and boosting methods like XGBoost in many cases using **default hyperparameters**, demonstrating robustness and reduced need for fine-tuning.

How the method works

Simple Overview:

1. Begin with a standard Random Forest.

2. Iteratively adjust sample weights to prioritize misclassified (hard) examples.
3. At prediction time, dynamically weight trees based on their performance on similar (neighboring) training examples.
4. Optionally remove low-importance samples/features to clean and compress the dataset.
5. Recover interpretability by analyzing top contributing trees per prediction.

Detailed Steps:

1. Sample Weighting

- Initialize all sample weights equally.
- Train a Random Forest using these weights and bootstrap sampling with weighted probabilities.
- Compute misclassification error using Youden's J statistic.
- Update sample weights to favor harder examples.
- Iterate until convergence or early stopping.

2. Model Weighting (Tree Weights)

- For a test point, find its nearest neighbors in the training set.
- Identify which trees performed best on those neighbors.
- Use only those trees with higher weights to make a personalized prediction.

3. Interpretability

- Rank trees by how often they were among the top-performing trees for neighbors.
- Provide final per-sample prediction as an interpretable aggregation of top trees.
- This allows end-users (e.g., clinicians) to inspect a handful of trees to understand the model's decision.

4. Sample and Feature Cleaning

- Low-importance samples and features are iteratively pruned.
- Reduces dataset size without degrading performance (and sometimes even improving it).

5. Evaluation

- Compared against CART, Random Forest, AdaBoost, and XGBoost on 15 binary classification datasets.
- ERF consistently outperformed vanilla RF and was on par or better than boosting methods, especially with default settings.