A Survey of Ensemble Learning Methods: Random Forest, AdaBoost, and XGBoost

Maxim Lavrenko
Department of Computer Science
Purdue University
`mlavrenk@purdue.edu`

December 6, 2024

Abstract

In recent years, ensemble learning techniques such as Bagging, AdaBoost, and Extreme Gradient Boosting (XGBoost) have become indispensable in machine learning workflows. Although these methods are often overshadowed by approaches inspired by human cognition, they continue to yield state-of-the-art performance in a variety of applied settings. In this paper, I examine the practical merits of these ensemble techniques by evaluating them on several real-world datasets. Beyond accuracy, I consider factors like computational overhead and resilience to noisy inputs. My goal is to provide a clearer picture of when and why these methods shine, and to highlight cases where their complexity may not pay off.

1 Introduction

Ensemble learning combines multiple predictive models (often called ”learners”) into a single, more robust predictor. By doing so, it seeks to mitigate the limitations of any one model and ultimately deliver better generalization. In this survey, I focus on three prominent ensemble methods—Bagging (exemplified by Random Forest), AdaBoost, and Gradient Boosting (specifically XGBoost)—to explore how each handles fundamental challenges like reducing variance, bias and overfitting.

What prompted this investigation was the recurrent observation of ensemble methods, particularly XGBoost, dominating machine learning competitions such as those on Kaggle. While browsing discussions about model selection, I came across a Reddit thread entitled *Why are ensemble models like XGBoost rarely used as primary methods or baselines in applied papers?*[1]. A single response encapsulated a widespread sentiment:

> ”Because in many cases they would outperform the model in the paper, with far less work and effort.”

This remark prompted me to take a closer look at these techniques and understand the conditions under which they excel. Instead of relying on assumption or reputation alone, I set out to assess their performance, computational efficiency, and resilience to challenging scenarios. Through a careful review of their theoretical underpinnings, coupled with empirical evaluations on diverse datasets, I attempt to shed light on how ensemble methods help us navigate the trade-offs inherent in machine learning.

2 Related Work

The concept of ensemble learning has evolved through a series of key contributions that reshaped how we combine multiple models to achieve more reliable predictions. One of the earliest and most influential pieces of this puzzle was Breiman’s 1996 paper, *Bagging Predictors* [1]. Breiman introduced the idea of using bootstrap samples to build

[1] `https://www.reddit.com/r/MachineLearning/comments/x1ildj/d_why_are_ensemble_models_like_xgboost_rarely/`

numerous versions of a base learner—often a decision tree—and then averaging their outputs. This simple step proved remarkably effective in taming models that tended to overfit, essentially smoothing out the rough edges caused by variance in the training data.

Around the same time, Freund and Schapire presented their AdaBoost algorithm (1997) [3], which took a different tack. Instead of treating each learner as an equally important contributor, AdaBoost zeroed in on the tough cases—the data points that earlier models had misclassified. By adjusting the sample weights so that "hard" examples demanded more attention, AdaBoost turned a sequence of relatively weak learners into a collectively strong one. This insight—that one can iteratively focus on errors—enriched the ensemble learning toolkit and changed how researchers thought about bias reduction and incremental improvement.

Friedman's 2001 work on Gradient Boosting [4] advanced these ideas further, generalizing boosting to operate like a gradient descent procedure over a chosen loss function. This shift in perspective allowed the method to adapt more readily to various tasks and complexities, ultimately leading to widespread adoption. Following in these footsteps, Chen and Guestrin's XGBoost (2016) [5] offered a highly optimized implementation that integrated second-order gradient information, explicit regularization terms, and parallelization strategies. XGBoost's success on large and challenging datasets, not to mention its frequent appearance atop leaderboards in machine learning competitions, demonstrated that boosting was no mere academic curiosity—it was a practical powerhouse.

But the story doesn't end there. Even AdaBoost, one of the older ensemble methods, continues to draw interest from a theoretical standpoint. Belanich and Ortiz [6, 7] have examined its convergence properties and highlighted lingering open questions, reminding us that established algorithms still have surprises up their sleeves.

Together, these foundational works and ongoing inquiries set the stage for the analysis in this survey. By building on decades of research—from bagging's variance reduction to AdaBoost's targeted error correction to the computational finesse of XGBoost—we gain a richer understanding of when and why certain ensemble methods excel and how we might refine them further.

3 Methodology

3.1 Bias-Variance Trade-off

Before digging into the details of these ensemble methods, I'd like to revisit a core idea that often comes up when we're trying to understand why certain approaches work and others don't: the bias-variance trade-off. I remember first learning about this concept and feeling as if I'd discovered a hidden compass for navigating model complexity. It's not always easy to follow, but it's helped me (and many others) make sense of why some models fail to generalize, while others hold up surprisingly well.

To put it simply, imagine you have a model that's too rigid—a straight-line approximation for something that's really more curved and nuanced. Such a model has a high bias: it consistently misses the mark by simplifying reality too much. Then consider the opposite extreme: a model that's so finely tuned to the training data that it practically memorizes every quirk and noise pattern. This one has high variance and will likely crumble the moment you show it new data.

The sweet spot lies somewhere between these two extremes, and finding it often feels like trying to balance on a narrow beam. Too much bias, and your predictions are just off in a systematic way. Too much variance, and you can't trust your model to perform outside its cozy training environment. Practitioners have wrestled with this tension for decades, sometimes stumbling on fixes by intuition, other times by trial and error.

Ensemble methods emerged as a kind of compromise. Instead of putting all your faith in a single model—whether it's too simple or too erratic—you combine several learners, hoping their collective wisdom leads to more stable performance. Some of these models might be a bit biased, others a bit too fancy, but when averaged or otherwise combined, they often yield something more reliable than any single contributor could achieve.

In the next sections, I'll show how Bagging, AdaBoost, and XGBoost each attempt to tame bias and variance in their own distinct ways.

3.2 Bagging

Bagging is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

The stability of the model can be quantified by the variance of the model. As discussed before, high variance means that the model is sensitive to the training data and will change significantly when trained on different subsets of the data, which means it's unstable. By aggregating the predictions of multiple unstable models, bagging hopes to cancel out the variance and reduce the overall variance of the model.

3.2.1 Random Forest

As said by Breiman in 2001 [2], Random Forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.

Below is pseudocode for a Random Forest algorithm [8]:

Algorithm 1 Random Forest

```
1: Input: A training set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, features $F$, and a number of trees in forest $\mathbf{B}$
2: function RandomForest($S, F$)
3:     $H \leftarrow \emptyset$
4:     for $i \in 1, \ldots, B$ do
5:         $S_i \leftarrow$ a bootstrap sample of $S$
6:         $h_i \leftarrow$ RandomizedTreeLearn($S_i, F$)
7:         $H \leftarrow H \cup \{h_i\}$
8:     end for
9:     return $H$
10: end function
11: function RandomizedTreeLearn($S, F$)
12:     At each node:
13:         $f \leftarrow$ a very small random subset of $F$
14:         Split on the best feature in $f$
15:     return The learned tree
16: end function
```

The idea is very simple. Instead of just aggregating multiple decision trees, we change the decision tree buidling process, to consider only a subset of features when selecting the best split at each node. This way, the trees are more diverse and less correlated, which helps to reduce the variance of the model.

3.3 Boosting

Boosting is another popular strategy for building ensemble models, but it approaches the problem differently than bagging does. Rather than training all learners independently and then averaging their predictions, boosting builds a sequence of models where each new model zeroes in on the mistakes made by the ones that came before it. Over time, this chain of weak learners transforms into a strong classifier that can handle patterns a single simple model would miss.

There's a useful way to think about this process in terms of bias and variance. If we consider variance to be a measure of how jittery or unstable a model is, we might think of bias as indicating how much crucial detail the model fails to capture. A model with high bias is too simplistic—it just doesn't "get" the complexity hidden in the data. Boosting tackles this by steadily refining the model, step by step, reducing bias and allowing the final classifier to pick up on the intricate patterns the initial weak learners overlooked.

3.3.1 AdaBoost

AdaBoost, introduced by Freund and Schapire in 1997 [3], is one of the earliest and most influential boosting algorithms. Its central insight is straightforward: identify which training examples are hardest to classify correctly, and then force the next model in the sequence to pay closer attention to those tough cases. Concretely, AdaBoost assigns higher weights to misclassified points, encouraging subsequent weak learners to focus on these tricky spots. Over many iterations, the algorithm builds a powerful predictor that, in principle, should be better at handling both easy and challenging examples.

The key idea behind AdaBoost is to assign exponentially increasing weights to the misclassified instances, and exponentially decreasing weights to the correctly classified instances. This way, the next weak learner focuses on the harder-to-classify data points, and the final strong classifier is a weighted sum of the weak learners.

Algorithm 2 AdaBoost

```
1: **Input:** Training data $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, number of iterations $T$, distribution $D$ over the $N$ examples, weak learner $h_t(x)$
2: **Output:** Final strong classifier $H(x)$
3: Initialize the weights as $w_i^{(1)} = D(i)$ for $i = 1, 2, \ldots, N$
4: **for** $t = 1$ to $T$ **do**
5:     Err = $\sum_{i=1}^{N} w_i^t \cdot \mathrm{I}(y_i \neq h_t(x_i))$
6:     Learn classifier $h_t(x)$ with weighted samples
7:     $\alpha_t = \frac{1}{2}\log(\frac{1-\mathrm{Err}}{\mathrm{Err}})$
8:     Update weights: $w_i^{(t+1)} = \frac{w_i^{(t)} \cdot \exp(-\alpha_t \cdot y_i \cdot h_t(x_i))}{\sum_{i=1}^{N} w_i^{(t+1)}}$
9: **end for**
10: Final model: $H(x) = \mathrm{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$
```

But if you think about it, if we're assignning exponentially increasing weights to the misclassified instances, wouldn't that lead to overfitting?

Despite that, AdaBoost has been shown to be a powerful algorithm that can achieve high accuracy on many different types of datasets, and the question of why AdaBoost works so well remains an open problem. In 2001, Breiman conjectured that AdaBoost was an random forest [2]. He argued that Adaboost, despite being a deterministic algorithm, behaves like a random forest because its weight updates on the training set appear to create distributions similar to those obtained through random weight selection. This observation is based on the empirical observation that the test set error of Adaboost often converges to an asymptotic value as more trees are added, suggesting a resistance to overfitting typically associated with random forests. [6]

In Section 4 I used the maximum depth of the weak learner to be 3, but it can actually be set to something more like 20, and have even better performance, which is contrary to the common belief that AdaBoost should use weak learners. Abraham J. Wyner et al. [9] paper's main point is that AdaBoost works well not in spite, but percisely because of its ability to interpolate the training data. That is, AdaBoost is able to fit the training data very well, and this is what allows it to generalize well to unseen data.

3.3.2 XGBoost

Gradient Boosting is another popular boosting algorithm that was introduced by Friedman in 2001 [4]. The algorithm works by training a sequence of weak learners, each of which learns to correct the errors of its predecessor.

XGBoost is a scalable and efficient implementation of Gradient Boosting that was introduced by Chen and Guestrin in 2016 [5]. The algorithm extends the Gradient Boosting framework by adding regularization and efficient parallel computation, making it a popular choice for machine learning competitions. XGBoost introduces several key technical contributions that improve upon traditional Gradient Boosting methods

The sparsity-aware split finding algorithm handles all sparsity patterns in a unified way. And what's even more impressive, is that exploits the sparsity to make the computation linear to a number of non-missing entries [5]. This is monumental, because unlike cherry picked datasets and simulated ones, real-world data often has missing values.

To combat overfitting, XGBoost adds a regularization term to the objective, so that each tree's complexity is penalized:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2,$$

where f is a tree with T leaves, and w are the weights of the leafs. The parameters γ and λ control the trade-off between model complexity and generalization, ensuring that trees do not become too deep or too simple [5].

On the systems side, XGBoost employs a cache-aware block structure for storing features, improving cache locality and parallelization. This makes it an extremely efficient and scalable algorithm [5].

In summary, XGBoost's explicit regularization, sparsity handling , and system-level optimizations, and more, provide substantial improvements in both speed and predictive power compared to classical gradient boosting approaches [5].

4 Experiments

4.1 Datasets

Inspired by ”Bagging Predictors” [1], I chose to use the following Datasetsts:

Dataset	Instances	Features	Task
Breast Cancer	699	9	Classification
Waveform	5,000	21	Classification
Letters	20,000	16	Classification
Bank Marketing	45,211	16	Classification
Adult	48,842	14	Classification
Wine Quality	4,898	11	Regression
Air Quality	9,358	15	Regression
Bike Sharing	17,379	13	Regression

Table 1: Datasets used for testing ensemble methods.

Description of each dataset can be found in the UCI Machine Learning Repository[2].

4.2 Experimental Setup

I used scikit-learn to implement test the performance of Random Forest, AdaBoost, and XGBoost on the datasets listed above. I used accuracy and F-1 score to evaluate the classification performance of the models, and mean squared error (MSE), as well as R-squared, to evaluate the regression performance of the models.

I also used a single decision tree as a baseline model to compare the performance of the ensemble methods.

4.3 Results

4.3.1 Classification Tasks

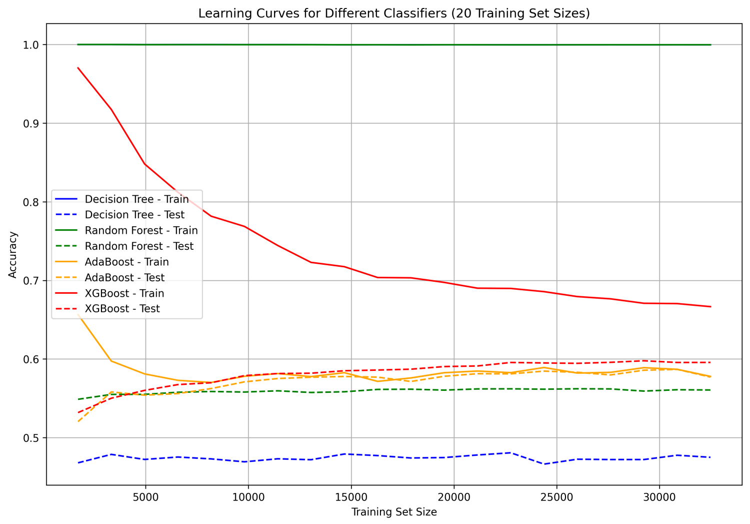

Figure 1: Learning curves for the Adult dataset.

Adult dataset: Unsurprisingly, a single decision tree and random forest fits the training data perfectly, achieving 100% train accuracy. More interstingly, AdaBoost’s train accuracy curve is much lower than that of XGBoost, which is

[2]https://archive.ics.uci.edu/ml/index.php

contrary to the common belief that AdaBoost overfits the training data, but makes sense given the recent research discussed in Section 3.3.1. Moreover, AdaBoost test accuracy converged to the train accuracy very fast, whereas XGBoost has a big gap, which suggests that XGBoost is more robust to overfitting than AdaBoost.

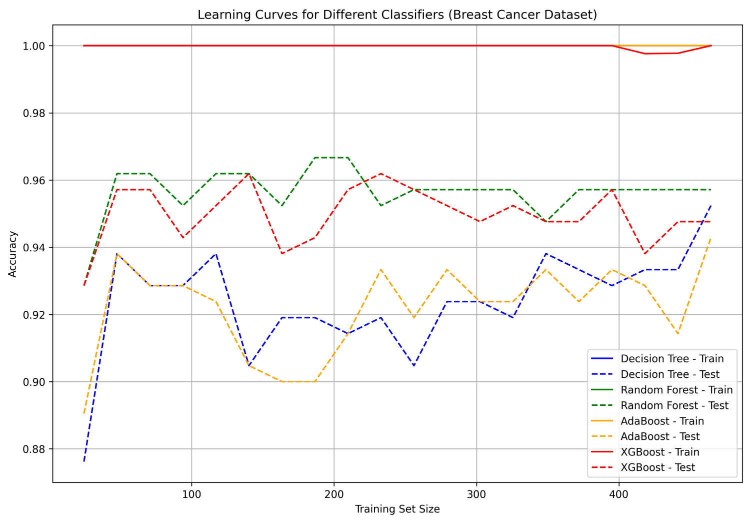

Figure 2: Learning curves for Breast Cancer dataset.

Breast cancer dataset: in this case, surprisingly all models have around 1.0 accuracy no matter the number of instances in the training set. This suggests that the dataset is very easy to classify, and the models can easily fit the training data. This is further supported by the fact that the test accuracy is also very high, which suggests that the models are generalizing well to unseen data.

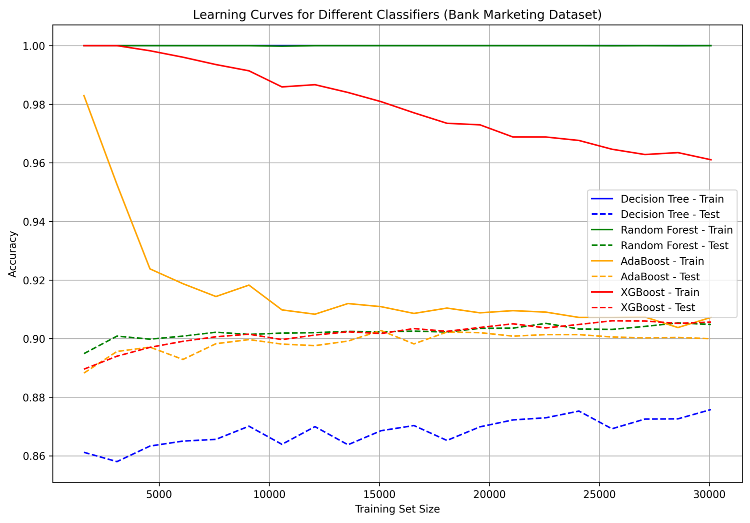

Figure 3: Learning curves for Bank Marketing dataset.

Bank Marketing dataset: a similar situation occurs in this dataset as it does in the adult dataset. The notable difference is that adaboost starts out with a large gap, and converges slower than it does for the adult dataset. Other than that, the test accuracy increases in a similar manner for all models (except the baseline single decision tree mode)

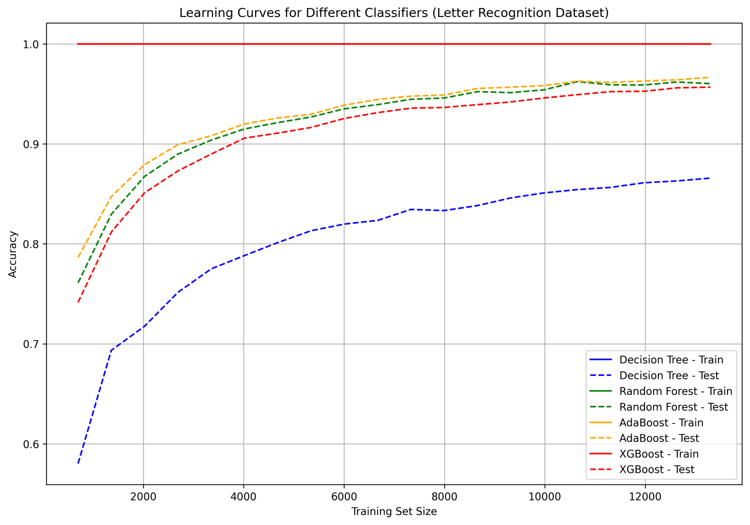

Figure 4: Learning curves for Letters dataset.

Letters dataset: the situation in this case is quite different. The test accuracy curves are still very similar for all 3 models, but they increase much faster than in the previous datasets, and the test curve is very smooth. This suggests that the models are generalizing well to unseen data, and the dataset is not very noisy. This is further supported by the fact that all models have 1.0 training accuracy.

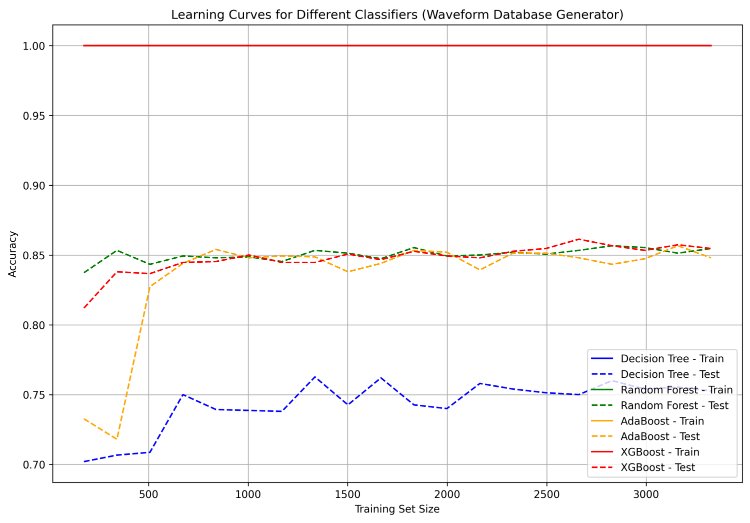

Figure 5: Learning curves for Waveform dataset.

Waveform dataset: the situation is unique from the previous ones. The training accuracy stays at 1.0, suggesting potential overfitting, and the test accuracy for all models don't improve much with size of the training set. Theoretically, we should have variance and bias problem reduced by the ensemble methods, but the constant gap between the training and test accuracy suggests presense of underlying error, which can be improved by using more features.

4.3.2 Regression Tasks

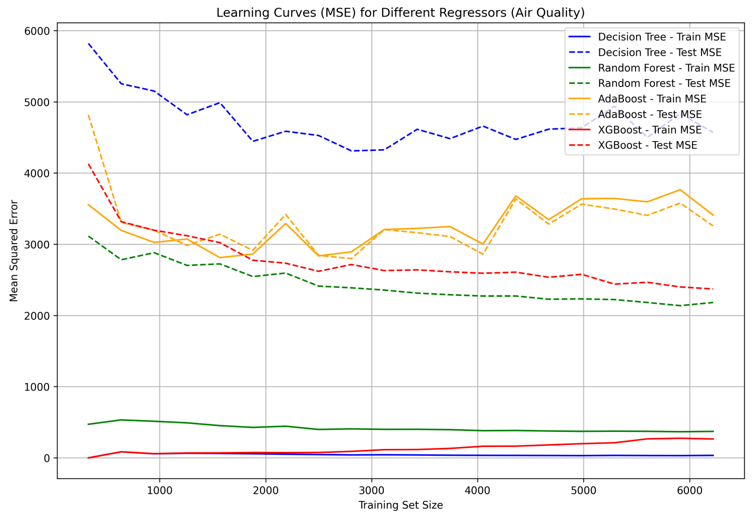

Figure 6: Learning curves (MSE) for Air Quality dataset.

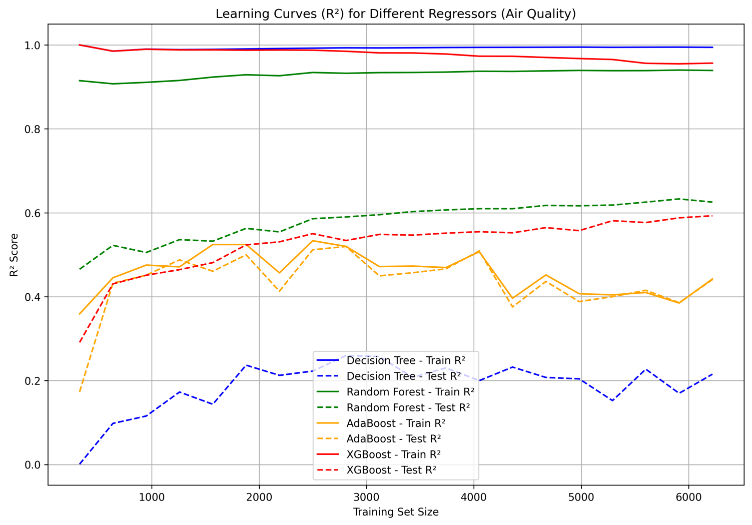

Figure 7: Learning curves (R-squared) for Air Quality dataset.

Air quality dataset: we have something interesting happening with AdaBoost here. Unlike single decision tree, random forest and XGBoost, AdaBoost has very similar training and testing performance, but it does generalize worse than the other models, as we see the test MSE going down and then up with a bigger training dataset.

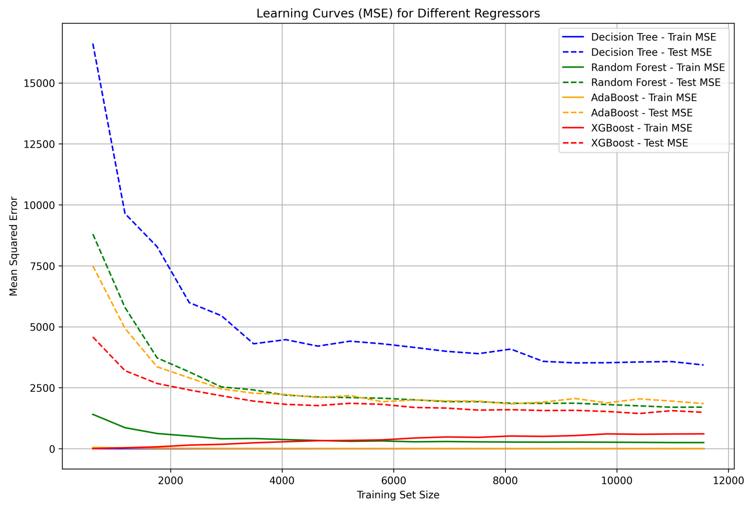

Figure 8: Learning curves (MSE) for Bike Sharing dataset.

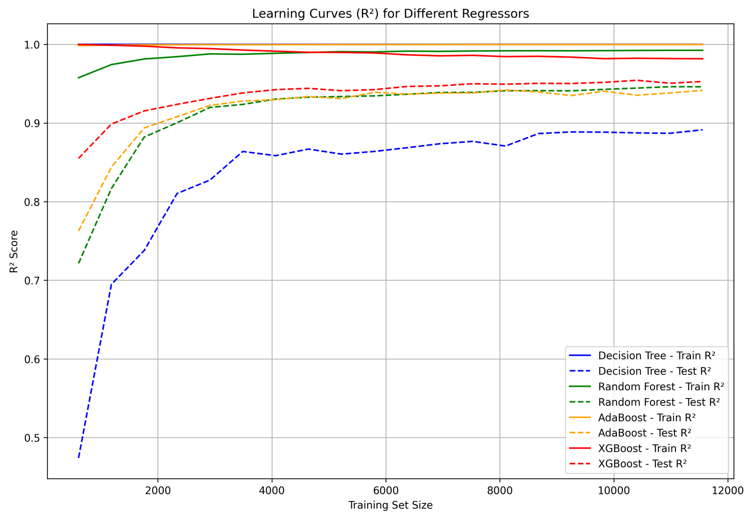

Figure 9: Learning curves (R-squared) for Bike Sharing dataset.

Bike Sharing dataset: similar to how the letters dataset was perfectly fitted on training data and had increasing accuracy on test data as the size of the training data is increased, the bike sharing dataset has very similar behavior. All the models have near 1.0 r-squared score on the training data, and the test r-squared score increases significantly with the size of the training data.

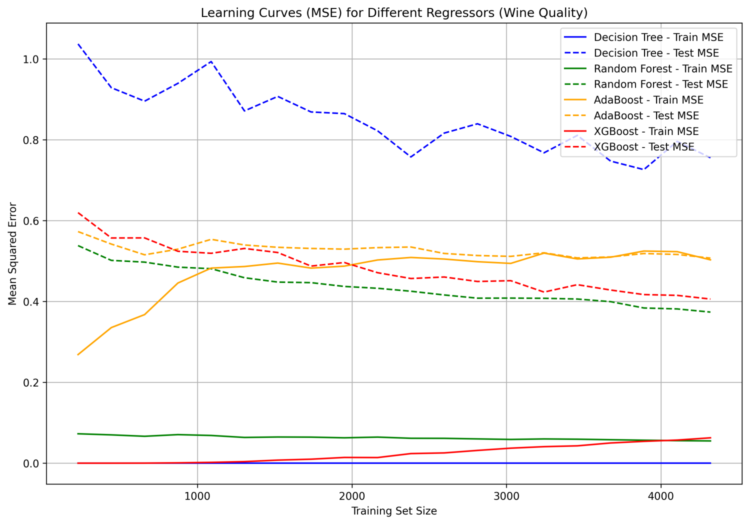

Figure 10: Learning curves (MSE) for Wine Quality dataset.

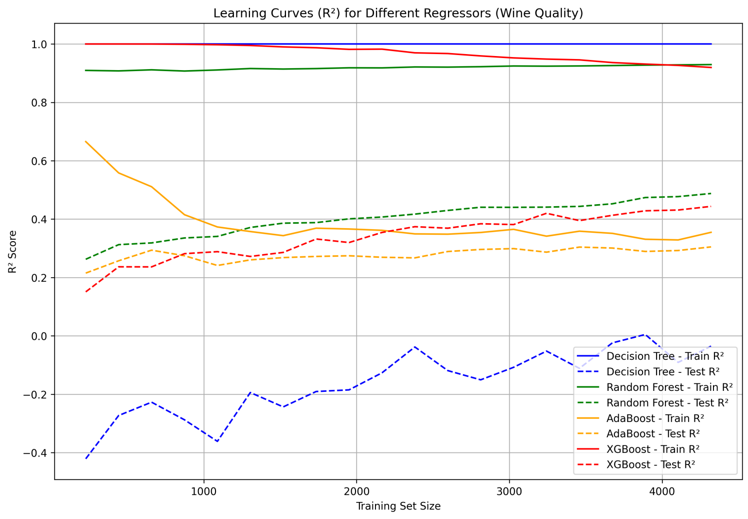

Figure 11: Learning curves (R-squared) for Wine Quality dataset.

Wine quality dataset: if the previous regression problem was very similar to the letters one, this one is very similar to the bank marketing one. AdaBoost converges pretty fast, whereas other models have a big gap between the training and test performanceand continue to improve in test performance as the size of the training data increases.

5 Discussion

Comparing the three methods discussed, we saw many interesting patterns in the experiments. In some classification tasks, we saw that AdaBoost often converged really fast, whereas XGBoost, and especially random forest, had high training score, and a big gap between the training and test score. In others, we saw AdaBoost and other models perform nearly perfect on the training dataset, leading to speculation that the dataset is very simple. It is difficult to draw concrete conclusions from this , since it's highly data dependent, and as we saw in 3.3.1, counterintuitive results can be observed.

The observations from the regression tasks were not so different. Again, we sometimes saw that AdaBoost's training R squared score started out lower than the other models, and converged faster, whereas the other models had a big gap between the training and test performance, and continued to improve in test performance as the size of the training data increased. And other times, we saw that all models had near perfect training performance, and the test performance increased with the size of the training data.

6 Conclusion

In conclusion, the experiments confirm that while all three ensemble methods improve upon single-tree baselines, their relative advantages are dataset-dependent. Understanding the nuances of each algorithm's design—particularly how they handle complexity, noise, and increasing training size—is crucial for practitioners aiming to select the best model for a given predictive task.

References

[1] Leo Breiman. *Bagging Predictors*. Machine Learning, 24(2):123–140, 1996.

[2] Leo Breiman. *Random Forests*. Machine Learning, 45(1):5–32, 2001.

[3] Yoav Freund and Robert E. Schapire. *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. Journal of Computer and System Sciences, 55(1):119–139, 1997.

[4] Jerome H. Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. Annals of Statistics, 29(5):1189–1232, 2001.

[5] Tianqi Chen and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794, 2016.

[6] Joshua Belanich and Luis E. Ortiz. *On the Convergence Properties of Optimal AdaBoost*. Department of Computer Science, Stanford University and Stony Brook University, January 4, 2023.

[7] Joshua Belanich and Luis E. Ortiz. *Some Open Problems in Optimal AdaBoost and Decision Stumps*. Department of Computer Science, Stanford University and Stony Brook University, 2024.

[8] University of Wisconsin–Madison. *Random Forests*. Available at: `https://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/ensembles/RandomForests.pdf`, accessed December 4, 2024.

[9] Abraham J. Wyner, Matthew Olson, Justin Bleich, and David Mease. *Explaining the Success of AdaBoost and Random Forests as Interpolating Classifiers*. Journal of Machine Learning Research, 18:1–33, 2017. Submitted 5/15; Revised 2/17; Published 5/17.