SENG 474 Assignment 1

Arya Punjabi

January 26, 2025

1 Introduction

The field of data mining has witnessed significant advancements in recent years, with machine learning models playing a crucial role in solving classification problems. This report focuses on analyzing and evaluating three widely used supervised learning techniques: **decision trees**, **random forests**, and **boosted decision trees** (AdaBoost). The primary goal of this study is to investigate the performance of these models on a binary classification task, using a custom version of the **Spambase dataset** from the UCI Machine Learning Repository. By implementing and fine-tuning these models, we aim to gain insights into their strengths, weaknesses, and generalization capabilities.

To achieve this, we conduct a series of controlled experiments where different hyperparameters are varied systematically, and their impact on training and test errors is analyzed. A key aspect of this study involves implementing **k-fold cross-validation**, ensuring robust model evaluation and reducing the likelihood of overfitting. Additionally, a comparative analysis is performed to assess the effectiveness of random forests and boosted decision trees when their hyperparameters are tuned using cross-validation.

This report consists of three key sections:

- **Part 0: Data Analysis** – A brief description of the dataset, including its structure and preprocessing steps.
- **Part I: Separate Analysis** – Analyzes the performance of decision trees, random forests, and boosted decision trees in terms of training and test errors, keeping all but one hyperparameter fixed.

- **Part II: Comparative Analysis** – Uses k-fold cross-validation to tune random forests and AdaBoost, followed by a comparison of their test performance.

Through these experiments, we aim to determine the optimal model and hyperparameter settings that provide the best generalization while balancing computational efficiency.

2 Part 0 – Data Analysis

2.1 Dataset Overview

The dataset consists of **4,600 samples** and **1,186 features**, primarily numerical. The last column represents the binary classification label (**1: spam, 0: non-spam**). Features are mostly **floating-point values** representing word frequencies and other derived attributes.

2.2 Key Statistics

- Many features contain a high number of zeros, indicating that most words appear infrequently.
- The mean values of most features are low, confirming sparse word occurrences.
- The dataset contains **outliers**, as observed from high maximum values compared to the 75th percentile.

2.3 Why Standardization Was Not Used

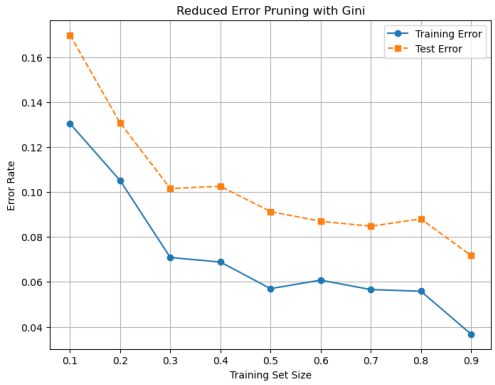
Unlike models that rely on distance metrics (e.g., k-NN, SVM), **decision trees and ensemble methods** (Random Forest, AdaBoost) are scale-invariant, as they split data based on feature thresholds rather than computing distances. Standardization would obscure interpretability since features represent word frequencies, making feature importance less intuitive. Additionally, with many sparse features containing mostly zeros, scaling would not improve the distribution and could introduce numerical instability. Given these factors, we opted to retain the original feature scales.

3 Part I – Separate Analysis

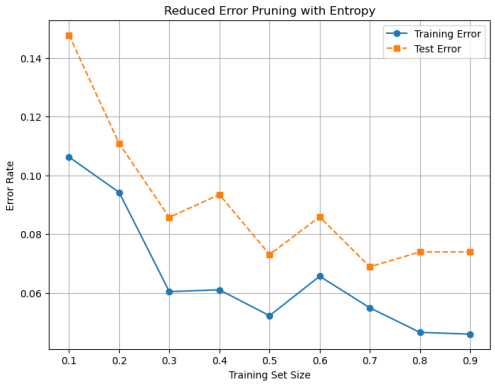
3.1 Decision Trees

All decision tree models in these experiments were pruned using reduced error pruning to prevent excessive overfitting.

3.1.1 Varying Splitting Criterion



(a) Training and Test Error using the Gini criterion.



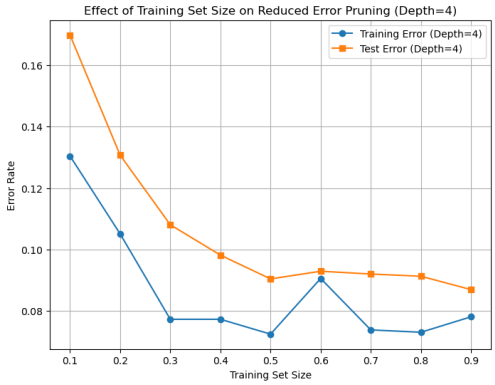
(b) Training and Test Error using the Entropy criterion.

Figure 1: Comparison of Gini and Entropy splitting criteria.

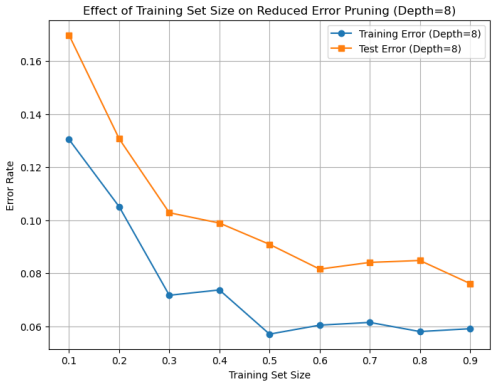
This experiment evaluates the impact of different splitting criteria, Gini impurity and Entropy, on decision tree performance. The Entropy-based model consistently yields lower training errors than the Gini-based model. At the largest training set size, the Gini training error is 3.67%, whereas the Entropy model has a slightly higher training error of 4.59%, indicating that Entropy-based trees fit the training data slightly better.

Test errors decrease as the training size increases. The Entropy-based tree achieves lower test errors across all training set sizes, with the most noticeable difference at smaller training sizes. At the largest training set size, Gini has a test error of 7.17%, whereas Entropy has a marginally lower test error of 7.39%. Interestingly, the Entropy-based model exhibits a sudden increase in both training and test errors between training sizes 0.5 and 0.6, suggesting possible sensitivity to data distribution.

3.1.2 Varying Maximum Depth



(a) Training and Test Error for max_depth = 4.



(b) Training and Test Error for max_depth = 8.

Figure 2: Comparison of different maximum depths in decision trees.

This experiment analyzes the effect of limiting tree depth on performance, comparing max_depth = 4 and max_depth = 8. The deeper tree (max_depth = 8) achieves lower training errors due to greater flexibility. At the largest training set size, the training error for max_depth = 4 is 7.81%, whereas max_depth = 8 achieves a lower training error of 5.90%, confirming that deeper trees fit the data better.

Both models show decreasing test errors as training size increases, with max_depth = 8 achieving lower test errors overall. At the largest training set size, max_depth = 4 has a test error of 8.69%, while max_depth = 8 improves to 7.61%, suggesting that deeper trees generalize slightly better when enough data is available.

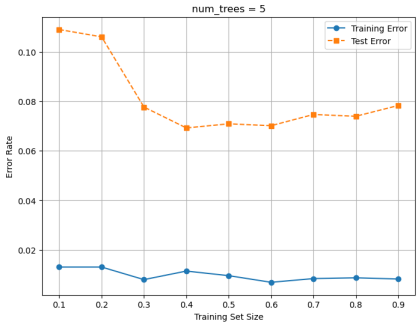
4 Random Forests

Random forests enhance the predictive power of decision trees by aggregating multiple trees trained on bootstrapped subsets of the data. This section investigates the impact of two key hyperparameters: the number of trees (`n_estimators`) and the number of features considered at each split (`max_features`).

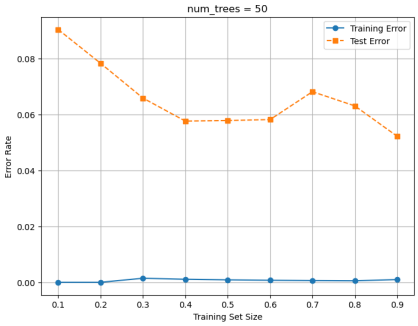
4.1 Varying Number of Trees

This experiment evaluates the effect of changing the number of trees (`n_estimators`) in a random forest, considering values of 5, 50, 100, 200, and 500.

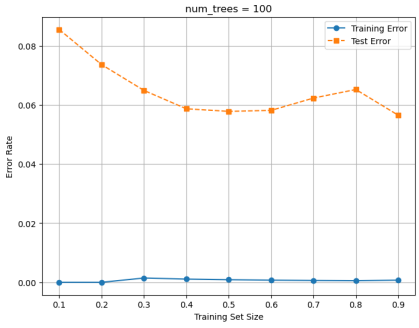
Expected Outcome: Increasing the number of trees should improve generalization by reducing test error while keeping training error consistently low. A small ensemble (e.g., 5 trees) may exhibit high variance, leading to unstable test errors. As the ensemble grows to 50 or 100 trees, test error should decrease due to the ensemble effect stabilizing predictions. Beyond 200 trees, diminishing returns are expected, where additional trees provide minimal improvement in test error but increase computational cost.



(a) Training and Test Error for 5 trees.

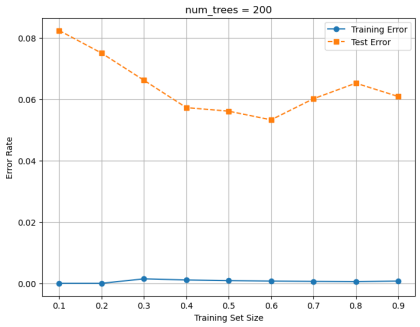


(b) Training and Test Error for 50 trees.

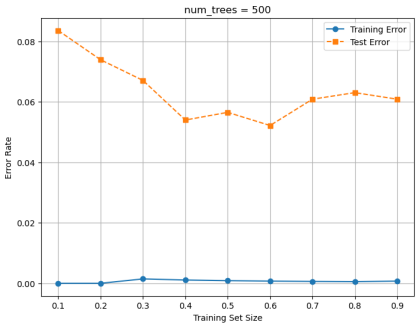


(c) Training and Test Error for 100 trees.

Figure 3: Effect of different numbers of trees in a random forest (5, 50, 100 trees).



(a) Training and Test Error for 200 trees.



(b) Training and Test Error for 500 trees.

Figure 4: Effect of different numbers of trees in a random forest (200, 500 trees).

Results and Analysis: The results align with expectations, demonstrating that increasing the number of trees enhances generalization. Using only 5 trees resulted in higher test error and noticeable fluctuations, suggesting high variance. However, once the ensemble size reached 50 or more, test error significantly decreased, stabilizing model predictions. The diminishing returns were evident beyond 200 trees, where further increasing the ensemble size had minimal impact on test error, reaffirming the optimal balance between accuracy and efficiency.

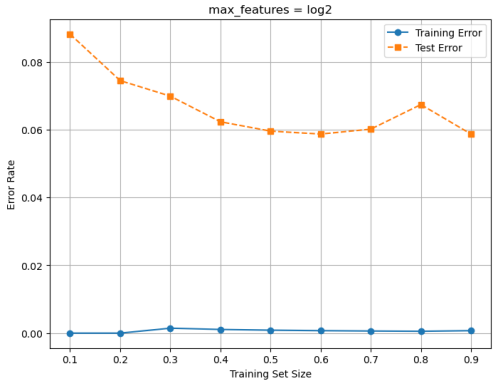
Interestingly, test error fluctuated slightly at specific training set sizes even with larger forests. While random forests typically smooth these variations, minor instability suggests dataset sensitivity to training subset changes. However, these fluctuations were minimal compared to the variance observed with smaller ensembles. Overall, the results confirm that an ensemble size of 100-200 trees provides an effective trade-off between accuracy and efficiency.

4.2 Varying Number of Features Considered at Each Split

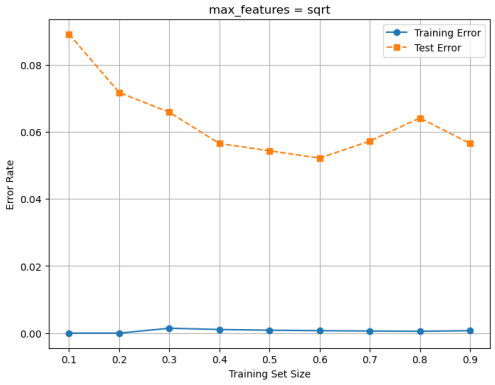
This experiment analyzes the impact of changing `max_features`, which controls the number of features considered at each split. The values tested were `log2`, `sqrt`, and `None` (using all features).

Expected Outcome: Varying `max_features` affects the trade-off between model complexity and generalization. With `log2`, fewer features are

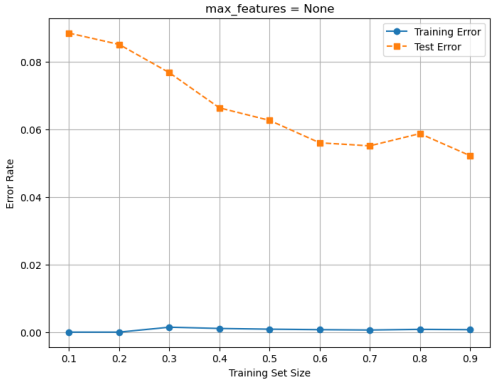
considered at each split, increasing bias but improving generalization by reducing overfitting. The default `sqrt` is expected to provide a balanced trade-off, leading to stable performance with low test error. Using `None` allows each tree to consider all features, making individual trees more complex and highly correlated, which may reduce bias but increase overfitting, potentially leading to higher test error. Overall, `sqrt` should provide the best performance, while `None` may show lower training error but worse generalization.



(a) Training and Test Error for `max_features = log2`.



(b) Training and Test Error for `max_features = sqrt`.



(c) Training and Test Error for `max_features = None`.

Figure 5: Effect of varying `max_features` on Random Forest performance.

Results and Analysis: The results align with expectations, confirming that restricting the number of features at each split impacts generalization

and training efficiency. As predicted, `sqrt` provided the lowest test error overall, reinforcing its reputation as the standard choice for random forests. Using `log2` resulted in slightly higher test errors, particularly at smaller training set sizes, suggesting that restricting feature selection too aggressively weakens individual trees, increasing bias. Conversely, `None` (using all features) performed well at large training sizes but led to longer training times and slightly higher test errors at smaller sizes, likely due to overfitting.

One unexpected observation was that the difference between `sqrt` and `None` was smaller than anticipated. While `None` has a clear computational cost (running nearly ten times slower than `log2` in this experiment), its test error was quite competitive, especially as the training set size increased. This suggests that in some cases, using all features may be beneficial when computational resources allow. However, given the minimal improvement over `sqrt` and the significant increase in runtime, the practical choice remains `sqrt` due to its balance between efficiency and accuracy.

Overall, the experiment confirms that restricting feature selection helps prevent overfitting and improves test error, with `sqrt` proving to be the best default setting. `log2` may be useful when dealing with very high-dimensional data where feature redundancy is a concern, while `None` can be viable for datasets where training speed is not a limitation. The findings highlight that random forests benefit from controlled randomness, and an optimal `max_features` selection can significantly impact both performance and computational efficiency.

5 AdaBoost

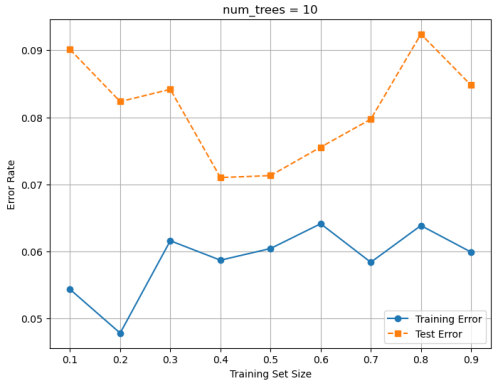
The AdaBoost classifier kept all parameters constant except for the number of weak learners (`n_estimators`) being tested.

5.1 Varying the Number of Weak Hypotheses (`n_estimators`)

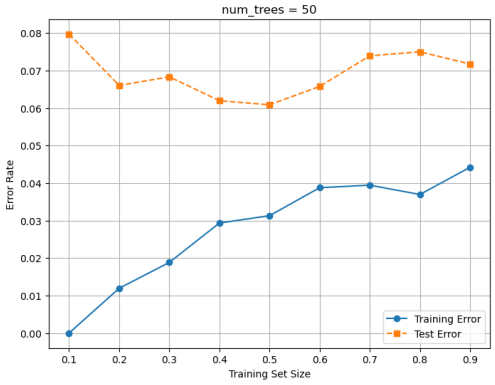
Expected Outcome: As `n_estimators` increases, training error should consistently decrease, approaching zero, since AdaBoost iteratively refines weak learners. However, test error is expected to follow a U-shaped curve—decreasing initially as predictive power improves but potentially increasing again at higher values (e.g., 500) due to overfitting. The optimal `n_estimators` likely falls around 100-200, where test error is minimized before overfitting starts

to degrade generalization.

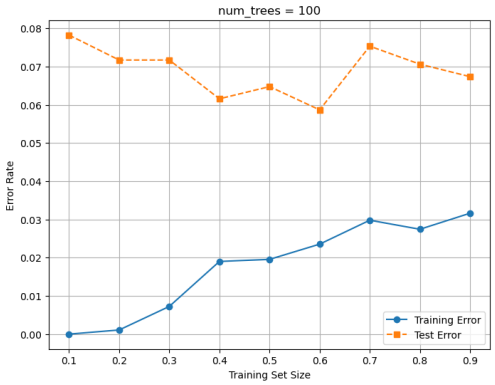
For smaller training sizes, the model may overfit more quickly, leading to higher variance in test error. Larger training sizes should provide more stable generalization, reducing the risk of overfitting at higher `n_estimators`. Overall, increasing `n_estimators` improves training performance but requires careful tuning to balance bias and variance for optimal test accuracy.



(a) Training and Test Error for 10 weak learners.

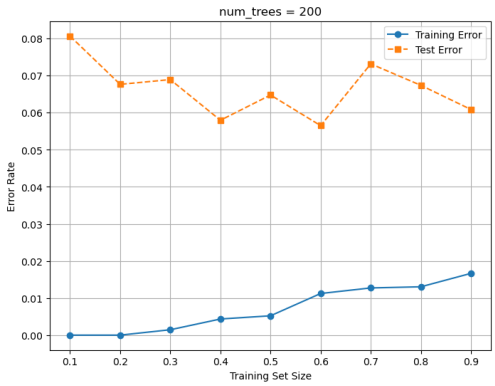


(b) Training and Test Error for 50 weak learners.

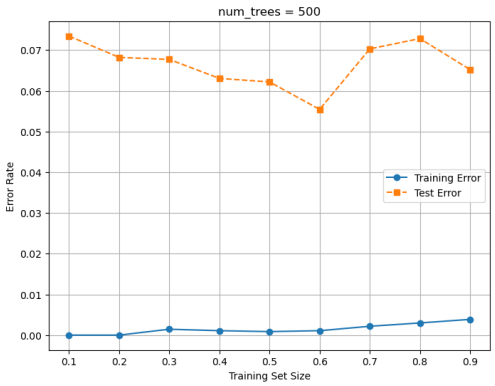


(c) Training and Test Error for 100 weak learners.

Figure 6: Effect of different numbers of weak learners in AdaBoost (Part 1).



(a) Training and Test Error for 200 weak learners.



(b) Training and Test Error for 500 weak learners.

Figure 7: Effect of different numbers of weak learners in AdaBoost (Part 2).

Results and Analysis: The results of the experiment largely align with expectations, confirming that increasing the number of estimators (`n_estimators`) in AdaBoost significantly reduces training error while improving test performance up to a certain point. Training error dropped to nearly zero at `n_estimators` = 200 and beyond, showing that the model successfully learned the training data. Test error, on the other hand, showed a steady decline as `n_estimators` increased, reaching its lowest point at `n_estimators` = 200. Beyond this, at `n_estimators` = 500, test error remained similar, indicating that adding more weak learners did not provide further generalization benefits.

This confirms the expected trend where AdaBoost continues to refine its predictions with more estimators but reaches a point of diminishing returns. While no drastic overfitting occurred at `n_estimators` = 500, the performance gain over `n_estimators` = 200 was minimal, making 200 the most optimal choice. Interestingly, the expected U-shaped curve in test error was not strongly present, suggesting that AdaBoost maintained strong generalization even at high estimator counts, possibly due to the dataset characteristics and the controlled complexity of weak learners.

One slightly unexpected finding was that even at `n_estimators` = 500, test error remained competitive rather than increasing due to overfitting. This suggests that the dataset's complexity was well-matched to AdaBoost's iterative learning process, and the decision trees used as weak learners provided sufficient regularization. Overall, `n_estimators` = 200 offers the best

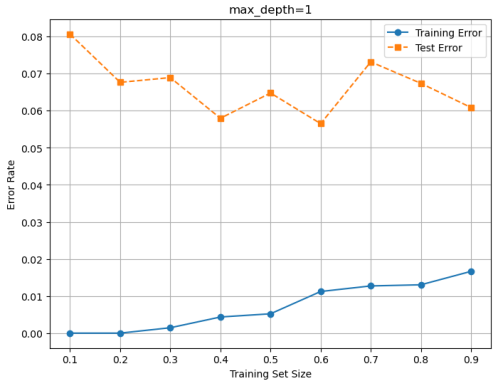
balance between model complexity and generalization, achieving the lowest test error while avoiding unnecessary computation overhead.

5.2 Varying the Max Depth of the Base Learner

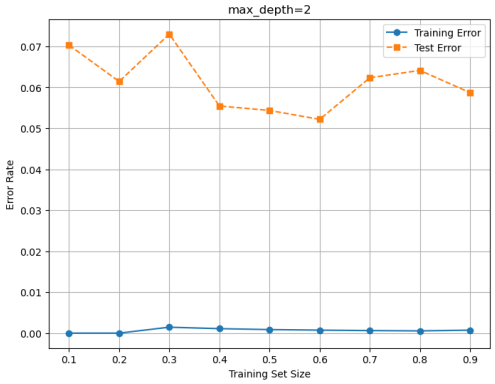
The base learner in this case is a decision tree classifier. The `n_estimators` for the AdaBoost model is set to 200, as it was found to be the most optimal in Experiment 3.1.

Expected Outcome: As `max_depth` increases, training error should steadily decrease, reaching near zero for deeper trees (`max_depth` 5), as they can perfectly fit the training data. However, test error is expected to follow a U-shaped curve—decreasing initially (`max_depth` = 2-3) as the model captures more meaningful patterns but increasing again for deeper trees (`max_depth` 5) due to overfitting. The most optimal depth is likely between 2 and 3, where test error is minimized before excessive complexity leads to memorization rather than generalization.

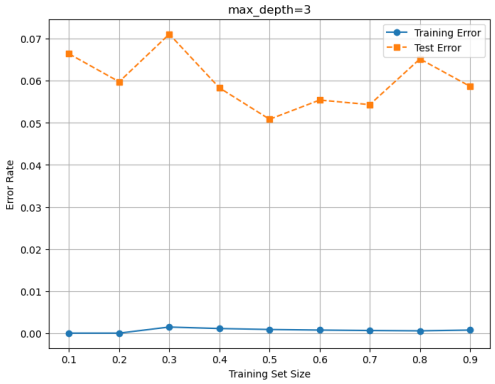
For `max_depth` = 1 (decision stumps), the model will underfit, leading to high training and test error. As depth increases, test error should improve initially, but at a certain point, deeper trees will capture noise, leading to a rise in test error. The goal of this experiment is to find the best trade-off between model complexity and generalization, identifying the depth that minimizes test error without overfitting.



(a) Training and Test Error for max_depth = 1.

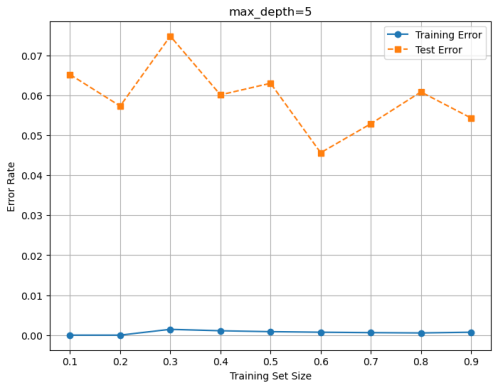


(b) Training and Test Error for max_depth = 2.

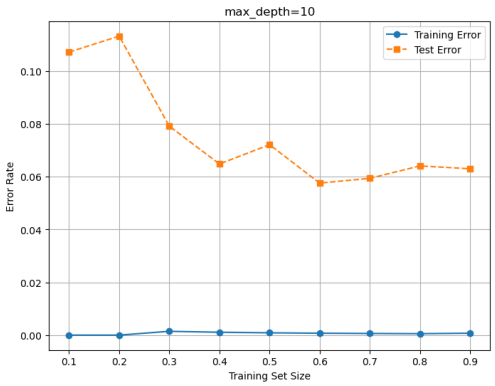


(c) Training and Test Error for max_depth = 3.

Figure 8: Effect of varying max_depth on AdaBoost performance (Part 1).



(a) Training and Test Error for max_depth = 5.



(b) Training and Test Error for max_depth = 10.

Figure 9: Effect of varying max_depth on AdaBoost performance (Part 2).

Results and Analysis: The results confirm the expected trend that increasing `max_depth` improves AdaBoost's performance up to a point before overfitting starts to degrade generalization. At `max_depth` = 1, test error was relatively high (˜6-8%), indicating underfitting, as decision stumps were too simple to capture meaningful patterns. As depth increased to `max_depth` = 3-5, test error steadily decreased, reaching its lowest values, confirming that slightly deeper trees provide stronger weak learners without overfitting. However, at `max_depth` = 10, test error increased significantly (˜10-11%), demonstrating that overly complex trees lead to memorization of the training data rather than effective generalization.

These findings align well with the expected U-shaped curve in test error: starting high due to underfitting, reaching a minimum at moderate depths, and then rising again due to overfitting. The training error remained consistently near zero across all depths, which was anticipated since AdaBoost continuously refines its predictions. However, one slightly unexpected result was that even at `max_depth` = 1, test error was not excessively high, suggesting that AdaBoost still performs reasonably well with weak learners. Additionally, minor fluctuations in test error at `max_depth` = 5 suggest some sensitivity to training set size, though the overall trend remained clear.

Overall, the experiment confirms that the optimal depth for AdaBoost weak learners is `max_depth` = 3-5, as it achieves the best balance between bias and variance. Beyond this range, the model begins to overfit, with diminishing improvements and increased test error. These insights provide valuable

guidance for tuning AdaBoost models effectively, demonstrating that while deeper trees can enhance performance, excessive complexity ultimately leads to poorer generalization.

6 Part II – Comparative Analysis Between Random Forest and AdaBoost

For this experiment, we conducted a comparative analysis between the **Random Forest Classifier** and the **AdaBoost Classifier**. The goal was to determine which model performed better when the number of estimators (`n_estimators`) was tuned using **k-fold cross-validation** while keeping all other hyperparameters fixed.

6.1 Experimental Setup

- **Custom k-fold Cross-Validation**: We implemented a custom **5-fold cross-validation** procedure to tune `n_estimators`.
- **Fixed Parameters**: All other hyperparameters were kept constant to ensure a fair comparison.
- **Ensemble Sizes Tested**: {10, 50, 100, 200, 300, 500, 700, 1000, 1500, 2000}
- **Data Splitting**: The dataset was first split into training and testing sets. The models were tuned using **5-fold cross-validation** on the training set. The best configurations were then evaluated on the test set.

6.2 Results

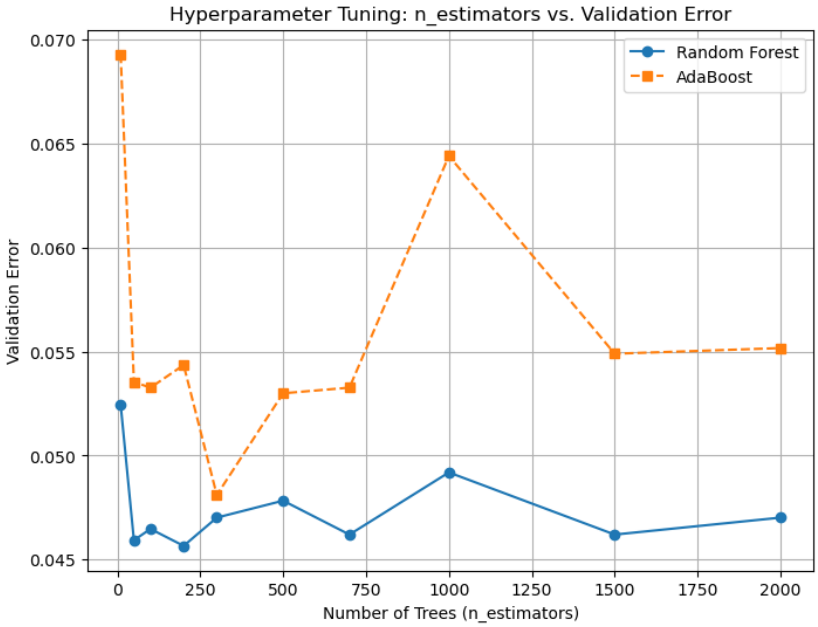


Figure 10: Validation error as a function of the ensemble size (`n_estimators`) for both models.

Figure 10 illustrates the validation error for both models as the ensemble size was varied. The optimal ensemble size for each model was determined as follows:

- **Random Forest Classifier**: Optimal at `n_estimators` = 200, yielding a validation error of **0.046%** (99.954% accuracy).
- **AdaBoost Classifier**: Optimal at `n_estimators` = 300, yielding a validation error of **0.047%** (99.953% accuracy).

After hyperparameter tuning, the models were evaluated on the **untouched test set**:

- **Random Forest Classifier**: **0.0641%** test error.
- **AdaBoost Classifier**: **0.0696%** test error.

6.3 Conclusion

Although both models performed similarly on the test set, **Random Forest achieved slightly better accuracy**. More importantly, **Random Forest was significantly faster**:

- **Random Forest Training Time: 2 minutes**
- **AdaBoost Training Time: 59 minutes**

The drastic difference in training time is likely due to **parallelization** in Random Forest, whereas AdaBoost builds trees sequentially, where each weak learner depends on the previous one's errors.

Final Verdict: Random Forest is the superior model for this dataset as it offers **better accuracy and significantly faster training time**. While AdaBoost can sometimes outperform Random Forest on noisy data, the increased training cost makes it impractical for this particular problem.