SENG 474 - Assignment 1

Evan Strasdin

February 2, 2025

Intro

We use the Spambase Augmented dataset - a modified version of the original Spambase dataset obtained from the UCI Machine Learning Repository. The dataset is designed for binary classification, with the goal of distinguishing between spam and non-spam emails. The augmented version includes additional features to capture interactions between words, providing a richer representation of the email content.

The original dataset contains 57 features, including 48 word frequencies, 6 character frequencies, and 3 features related to capital letter sequences. The augmented dataset extends this by adding 1,128 new features, each representing the sum of frequencies for every pair of the 48 words. This brings the total number of features to 1,185. These pairwise interactions are particularly valuable for identifying co-occurrence patterns of words, which can be indicative of spam content. For example, the presence of certain word pairs like "free" and "offer" might significantly increase the likelihood of an email being classified as spam.

The dataset was shuffled to ensure diversity in the training and test sets, which were split into 80% training and 20% test data. The inclusion of pairwise word-frequency sums aims to capture more complex patterns in the data, making it well-suited for evaluating our classifiers: decision trees, random forests, and boosted decision trees. These models were chosen for their ability to handle high-dimensional data and their effectiveness in capturing both linear and non-linear relationships. Decision trees, for instance, can naturally handle interactions between features, while random forests and boosted decision trees leverage ensemble methods to improve predictive performance and reduce overfitting.

To further enhance the robustness of our analysis, we also performed feature scaling and normalization to ensure that all features contribute equally to the model training process. This step is crucial given the wide range of values in the dataset, from word frequencies to capital letter sequences.

We will evaluate the performance of the aforementioned models—decision trees, random forests, and boosted decision trees—across various values of key hyperparameters. Rather than identifying optimal configurations, our goal is to observe how changes in hyperparameters, such as tree depth, number of trees, and learning rate, impact model performance. This approach allows us to understand the sensitivity of each model to these parameters and provides insights into their behavior under different settings.

Additionally, we will perform k-fold cross-validation to assess the consistency and generalizability of the random forest and boosted decision tree models. By evaluating error, we aim to compare the models' effectiveness in distinguishing between spam and non-spam emails. This analysis will highlight the trade-offs and strengths of each approach, offering a clearer picture of their suitability for the task without focusing on hyperparameter optimization.

Part 1: Separate Analysis

1.1 - Decisions trees

The decision tree implementation in the code combines explicit parameter settings with scikit-learn's default values to create a comprehensive model configuration. The code explicitly sets `random_state=42` to ensure reproducible results across different runs by fixing the random seed for internal processes like handling split ties. For the baseline decision tree, several default parameters remain unchanged, including `max_depth=None` which allows unrestricted tree growth until leaves are pure or other stopping conditions are met.

Parameter Configuration The baseline configuration relies on fundamental default settings that shape the tree's behavior. The Gini impurity measure evaluates splits, while the "best" splitter algorithm selects optimal splits at each node. Internal nodes require a minimum of two samples to split (`min_samples_split=2`), and leaves can form with just one sample (`min_samples_leaf=1`). Initially, no cost-complexity pruning is applied (`ccp_alpha=0.0`), though this parameter becomes crucial in later optimization stages.

Model Performance Analysis The baseline decision tree exhibits classic overfitting behavior, achieving perfect accuracy (0.0 error) on the training set while showing significantly higher error (0.088) on the test set. This performance gap demonstrates the model's tendency to memorize training data rather than learn generalizable patterns.

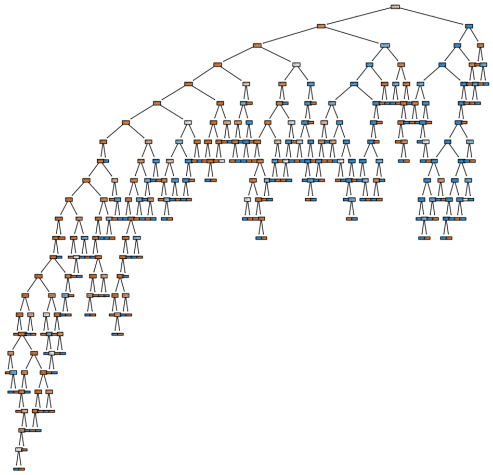

Figure 1: Visualization of the baseline decision tree.

Depth Impact Tree depth significantly influences model performance. Starting with a high error of 0.175 at depth 1, the training error rapidly decreases until depth 5, then gradually approaches zero beyond depth 15. Test error follows a different pattern, initially dropping sharply to 0.09 at depth 5, then stabilizing around 0.075-0.08 between depths 13-15, with slight degradation for deeper trees.

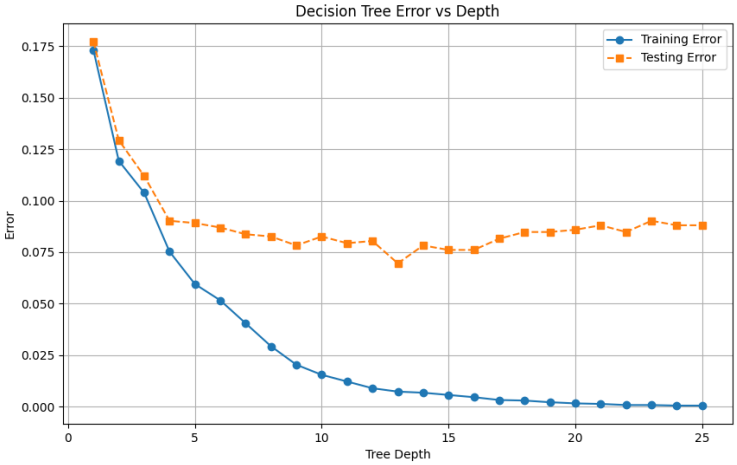

Figure 2: Training and test error as a function of tree depth.

Cost-Complexity Pruning Pruning strength (alpha) provides another dimension of control over model complexity. Training error remains near zero for small alpha values ($<$0.0005) before steadily increasing, with a sharp rise around alpha=0.0125. Test error reaches its minimum (0.07) at alpha=0.0007, followed by gradual deterioration and eventual stabilization around 0.13 for larger alpha values.

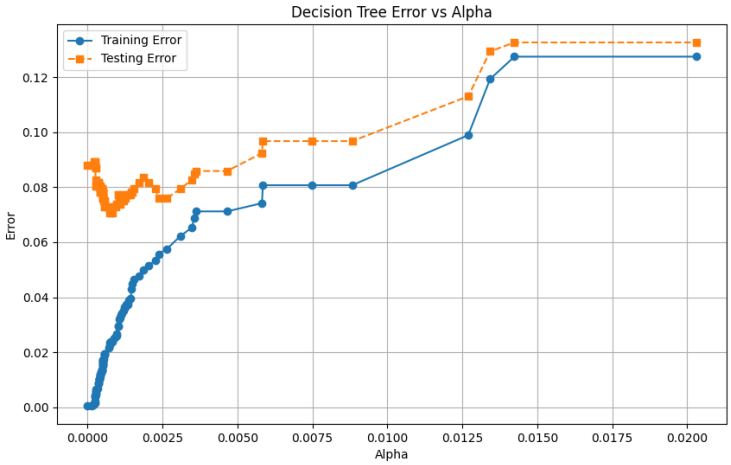

Figure 4: Training and test error as a function of alpha (pruning strength).

Learning Dynamics The learning curves reveal interesting patterns in model behavior across different training set sizes. While training error consistently remains near zero, demonstrating the model's perfect memorization capability, test error shows more complex behavior. Starting high at 0.12 with limited data, test performance becomes stable after using 40% of the training data, ultimately achieving its best performance of 0.08 with the full dataset.

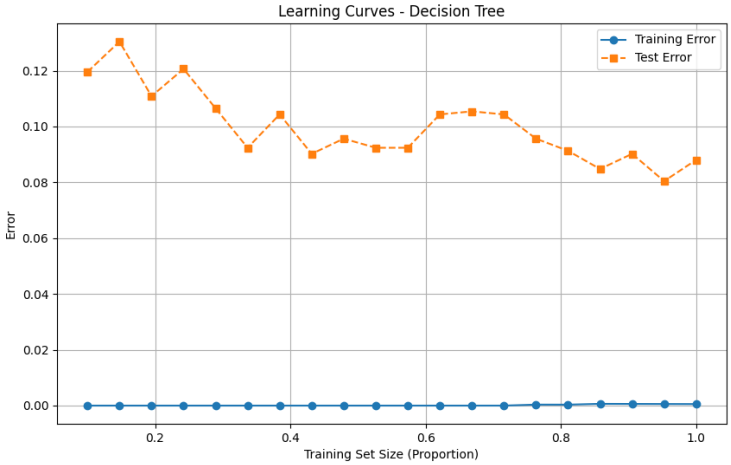

Figure 5: Training and test error as a function of training set size.

Conclusion The decision tree shows clear signs of overfitting across all experiments, with near-zero training error in most cases. The optimal depth appears to be around 13-15 levels, where test error stabilizes around $7.5 - 8\%$. Cost-complexity pruning achieves best results with alpha 0.0007, yielding similar test performance. The learning curves reveal that while more data generally helps, the improvement is modest and noisy, suggesting the model's fundamental limitations in generalizing to unseen data. The consistent gap between training and test error indicates that additional techniques beyond pruning might be needed for better generalization.

1.2 - Random Forests

The random forest implementation combines scikit-learn's default parameters with specific configurations to create a robust classification model. The code explicitly sets `random_state=42` to ensure reproducible results across different runs, while maintaining default values for most other parameters, including 100 trees (`n_estimators=100`) and automatic feature selection (`max_features="auto"`).

Parameter Configuration The baseline model utilizes fundamental settings that define the forest's behavior. It employs the Gini impurity measure for split evaluation and uses bootstrap sampling for tree construction. Each internal node requires a minimum of two samples to split, and leaves can form with just one sample. These defaults establish a foundation for subsequent performance analysis and optimization.

Model Performance Analysis The baseline random forest demonstrates impressive performance characteristics. With a training error of 0.0005, it achieves near-perfect accuracy on the training set. The test error of 0.0652, while higher, remains relatively low, showcasing the model's strong generalization capabilities despite the indication of some overfitting.

Forest Size Impact The number of trees significantly influences model performance. Training error starts at approximately 0.04 with a single tree and drops sharply in the first 10 trees, eventually stabilizing around 0.001 after 20 trees. Test error follows a similar pattern, beginning at 0.115, decreasing rapidly to 0.07 with 10 trees, and stabilizing around 0.063-0.065 after 20 trees.

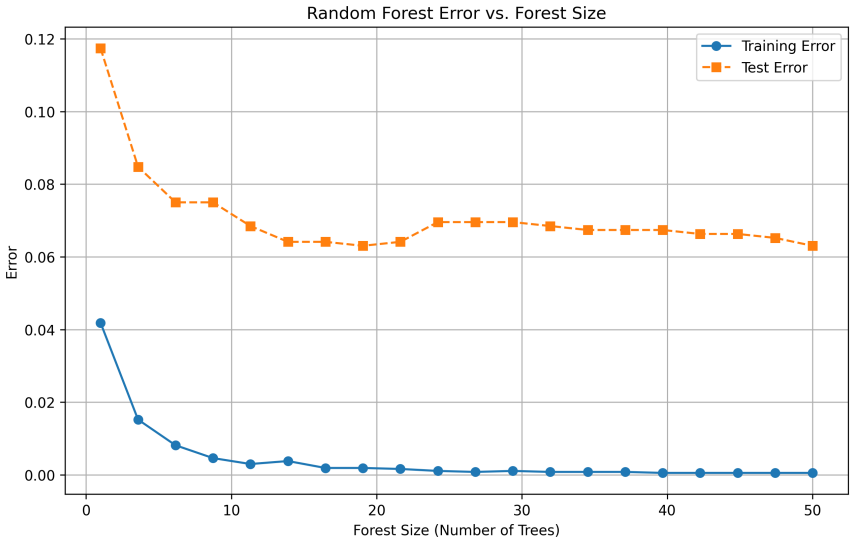

Figure 1: Training and test error as a function of forest size (number of trees).

Feature Subset Analysis The model shows remarkable stability across different feature subset sizes. Training error remains consistently near zero (˜0.001), while test error fluctuates between 0.055-0.063 without showing a clear optimal feature subset size. This stability suggests robust feature handling capabilities.

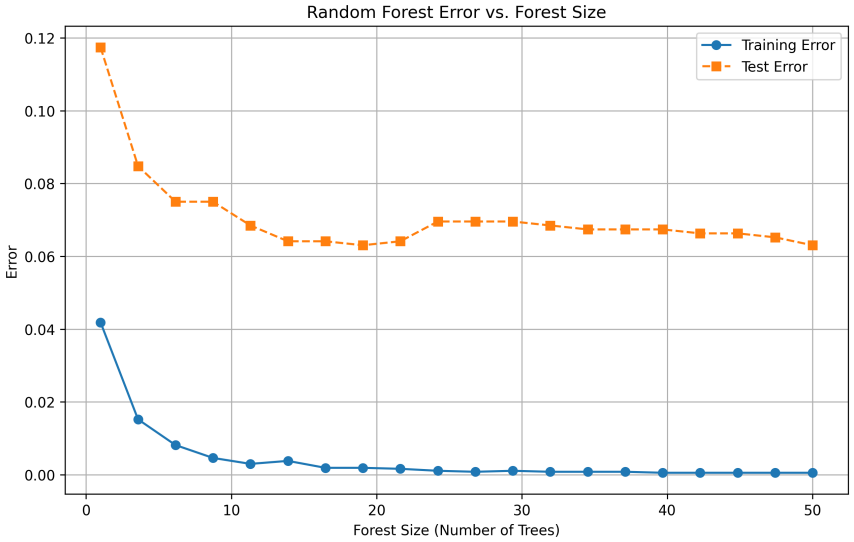

Figure 2: Training and test error as a function of feature subset size.

Learning Dynamics The learning curves reveal interesting patterns in model behavior across different training set sizes. Training error maintains exceptional stability near zero, while test error shows more dramatic changes. Starting at 0.11 with limited data, test performance improves sharply until reaching about 40% of the training data, then gradually stabilizes around 0.065 with the full dataset.

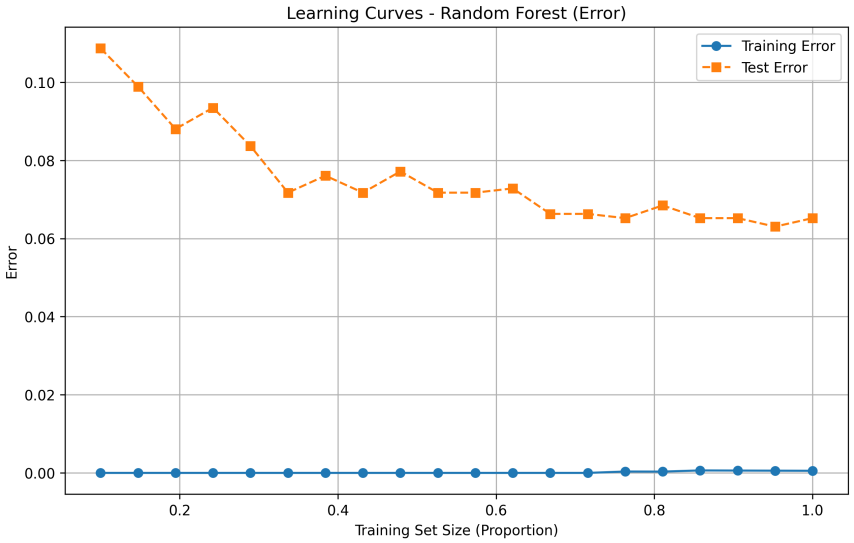

Figure 3: Training and test error as a function of training set size.

Conclusion The random forest demonstrates robust performance across different configurations. The model achieves optimal performance with around 20 trees, showing minimal improvement thereafter. Feature subset size has surprisingly little impact on performance, suggesting the model's robustness to feature selection. The learning curves indicate that 40% of the training data captures most of the learnable patterns, though performance continues to improve slightly with more data. The consistent near-zero training error across all experiments suggests some overfitting, but the stable test error around 6.5% demonstrates good generalization ability.

1.3 - Boosted Decision Trees

The AdaBoost implementation combines specific configurations with scikit-learn's default parameters to create an effective ensemble model. The code sets `random_state=42` for reproducibility and uses decision stumps as weak learners, with 50 estimators and a learning rate of 1.0 as default settings.

Model Performance Analysis The baseline AdaBoost model demonstrates balanced performance characteristics, achieving a training error of 5.33% and a test error of 7.93%. This relatively small gap between training and test performance suggests good generalization capabilities without severe overfitting.

Impact of Weak Learners The number of iterations significantly influences model performance. Training error begins at approximately 17% and shows a sharp decrease in the first 10 iterations, followed by gradual improvement until stabilizing around 5% after 60 iterations. Test error follows a similar pattern, starting at 17.5% and stabilizing around 8% after 40 iterations.

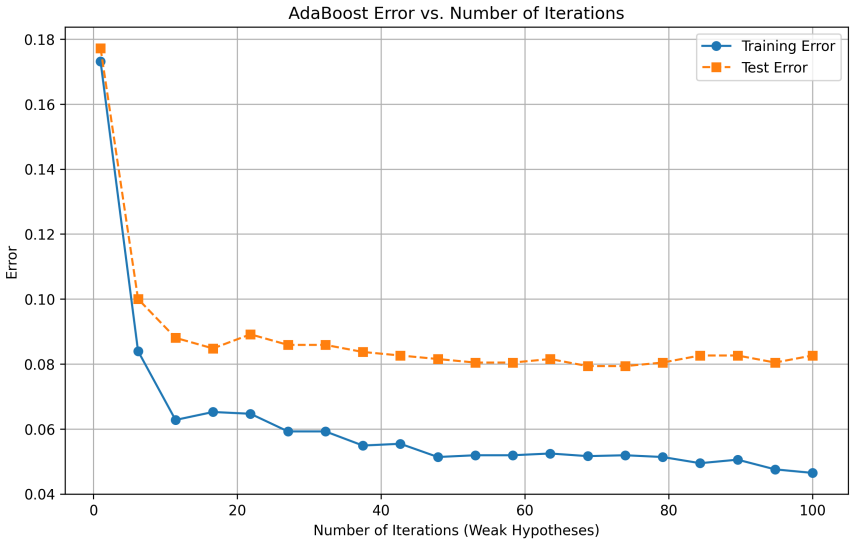

Figure 1: Training and test error as a function of the number of iterations.

Stump Depth Analysis The depth of weak learners shows interesting effects on model performance. Training error starts at 5.3% with depth-1 stumps and rapidly decreases to near-zero at depth 4. Test error begins at 7.9% and improves to around 6% at depth 4, maintaining stable performance between 5.8-6.3% for depths 4-10.

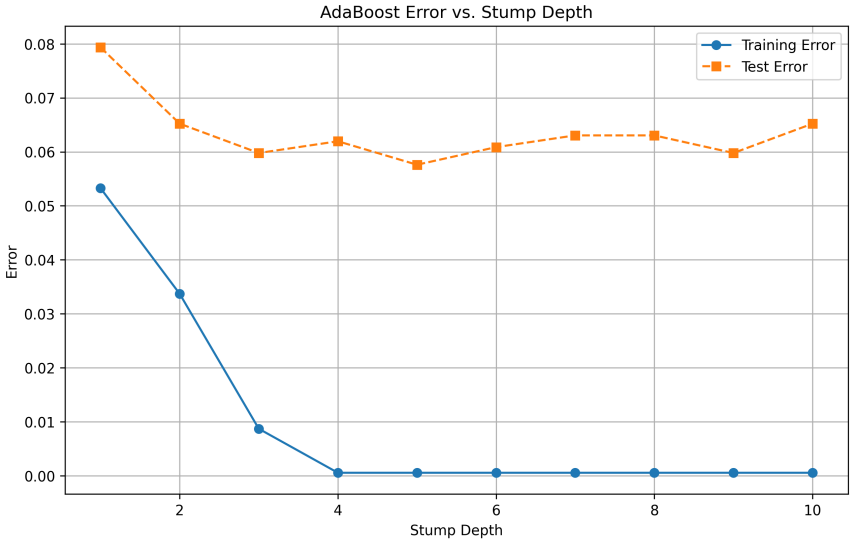

Figure 2: Training and test error as a function of stump depth.

Learning Dynamics The learning curves reveal unusual patterns. Training error starts very low (˜1%) with small datasets and increases steadily to 5.5% with more data. Test error shows high variability with small training sets, fluctuating between 8-9.5%, before stabilizing around 8% with the full dataset.

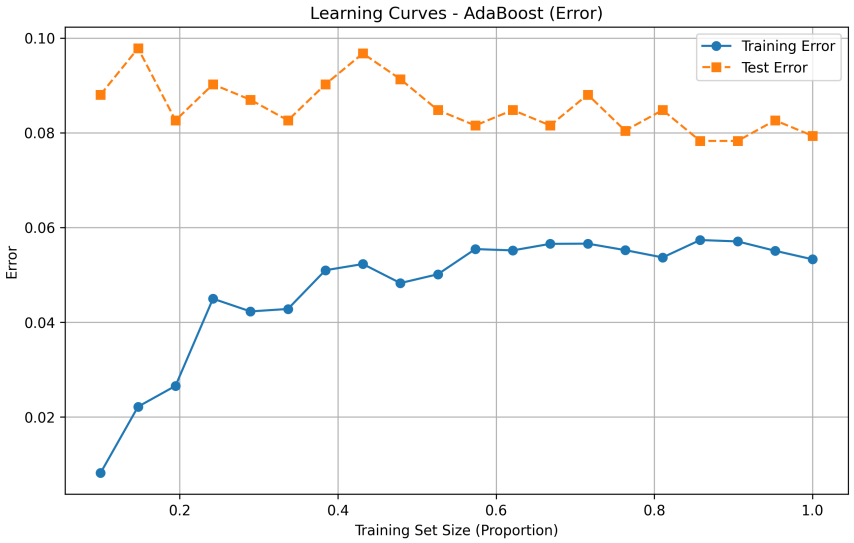

Figure 3: Training and test error as a function of training set size.

Conclusion The AdaBoost model shows interesting behavior across different parameters. The optimal configuration appears to be using depth-4 trees with around 40-60 iterations, as further complexity yields minimal improvements. The learning curves reveal that the model achieves stable performance with about 60% of the training data. Unlike typical machine learning models, the training error actually increases with more data, suggesting the model maintains good generalization by avoiding overfitting on larger datasets.

Part 2: Comparative Analysis

The k-fold cross-validation implementation is a robust approach to model evaluation that partitions the data into k equal-sized folds. The code includes essential features such as random shuffling of indices for unbiased fold creation, proper handling of the final fold's size to account for non-divisible datasets, and a clear error computation using 0-1 loss for classification tasks. The function accepts flexible inputs including numpy arrays and pandas DataFrames, making it versatile for different data formats. It returns both individual fold errors for detailed analysis and the average error across all folds, providing a comprehensive view of model performance. The implementation also ensures reproducibility through an optional `random_state` parameter, with default `42`.

2.1 - K-Fold Cross Validation with Random Forests

The random forest cross-validation analysis provides a comprehensive evaluation of model performance across different ensemble sizes, employing a systematic approach to understand generalization capabilities and optimal configuration.

Experimental Setup The analysis utilizes a Random Forest Classifier with carefully selected parameters for robust evaluation. The implementation employs 5-fold cross-validation to ensure reliable error estimation, testing ensemble sizes ranging from 10 to 150 trees in increments of 10. The model maintains consistency through a fixed random seed (42) and uses the standard square root of features for split decisions based on Gini impurity.

Performance Analysis The cross-validation results reveal distinct patterns in error behavior across different ensemble sizes. Starting with a relatively high error of 5.27% with 10 trees, the model shows dramatic improvement in its initial phase. A sharp error reduction occurs up to 40 trees, reaching 4.62%, followed by more gradual improvements until reaching 110 trees.

The error rate stabilizes between 4.4-4.5% for larger ensembles, with optimal performance achieved at both 110 and 140 trees, yielding a 4.40% error rate. This pattern suggests a clear point of diminishing returns in ensemble size scaling.

Fold-Level Performance Individual fold analysis demonstrates remarkable consistency across different data partitions. The performance variation typically remains within ±0.5% across folds, indicating robust generalization capabilities. Notable extremes include:

- Best performance: 3.53% error (fold 3, 140 trees)
- Worst performance: 5.57% error (fold 1, 10 trees)

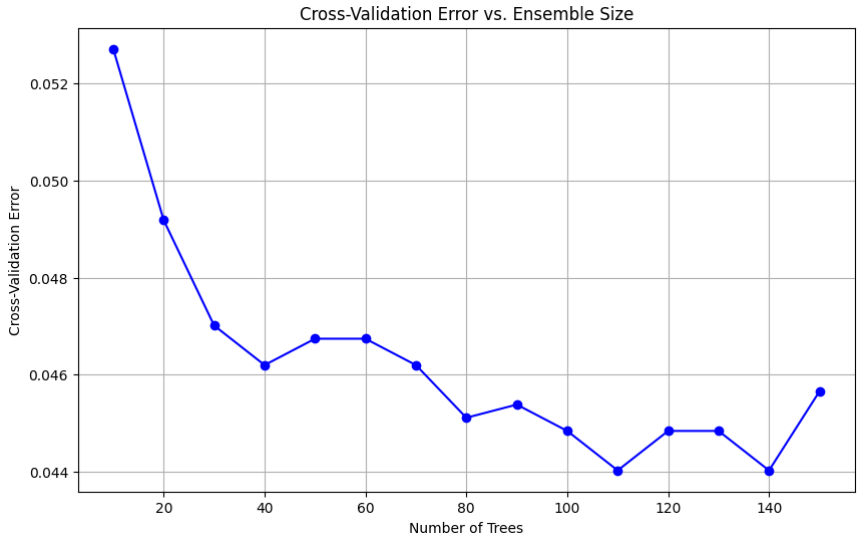

Figure 2.1: Cross-validation error as a function of ensemble size (number of trees).

Key Findings The analysis reveals several crucial insights about random forest behavior:

1. **Initial Improvement**: Substantial error reduction occurs in the first 40 trees, demonstrating the importance of minimum ensemble size.
2. **Optimization Point**: The model achieves optimal performance at 110 trees, suggesting this as an ideal configuration point.
3. **Stability**: Consistent performance across folds indicates reliable generalization capabilities.
4. **Efficiency Threshold**: Minimal improvements beyond 110 trees suggest this as an efficient cutoff point for ensemble size.

Conclusion The cross-validation analysis establishes 110 trees as the optimal ensemble size, achieving a 4.40% cross-validation error while maintaining computational efficiency. This configuration represents an effective balance between model complexity and generalization ability, with performance stability across different data partitions confirming the model's reliability. The clear

diminishing returns pattern beyond this point provides practical guidance for model deployment, suggesting that larger ensembles offer minimal benefits while increasing computational overhead.

This comprehensive evaluation demonstrates that random forests can achieve robust and stable performance with moderate ensemble sizes, providing valuable insights for practical implementation in similar classification tasks.

2.2 - K-Fold Cross Validation with Boosted Decision Trees

The AdaBoost cross-validation analysis provides a detailed examination of model performance using decision stumps as weak learners, systematically evaluating ensemble sizes to determine optimal configuration and generalization capabilities.

Experimental Setup The analysis employs an AdaBoost Classifier with specific parameter choices for thorough evaluation. The implementation uses 5-fold cross-validation with ensemble sizes ranging from 10 to 150 stumps in 10-stump increments. The model maintains consistency through a fixed random seed (42) and uses depth-1 decision trees (stumps) as weak learners with the default learning rate of 1.0.

Performance Analysis The cross-validation results demonstrate clear patterns in error behavior across ensemble sizes. The model begins with a 7.17% error rate using 10 stumps and shows significant improvement in its initial phase. A marked error reduction occurs up to 40 stumps, reaching 5.98%, followed by more gradual improvements until 110 stumps.

The error rate stabilizes between 5.52-5.60% for larger ensembles, with optimal performance achieved at both 110 and 130 stumps, yielding a 5.52% error rate. This pattern clearly indicates a point of diminishing returns in ensemble size scaling.

Fold-Level Performance Individual fold analysis reveals more substantial variation compared to random forests, with typical differences of $\pm 1\%$ across folds. Notable performance extremes include:

- Best performance: 4.48% error (fold 4, 110-130 stumps)
- Worst performance: 7.88% error (fold 5, 10 stumps)

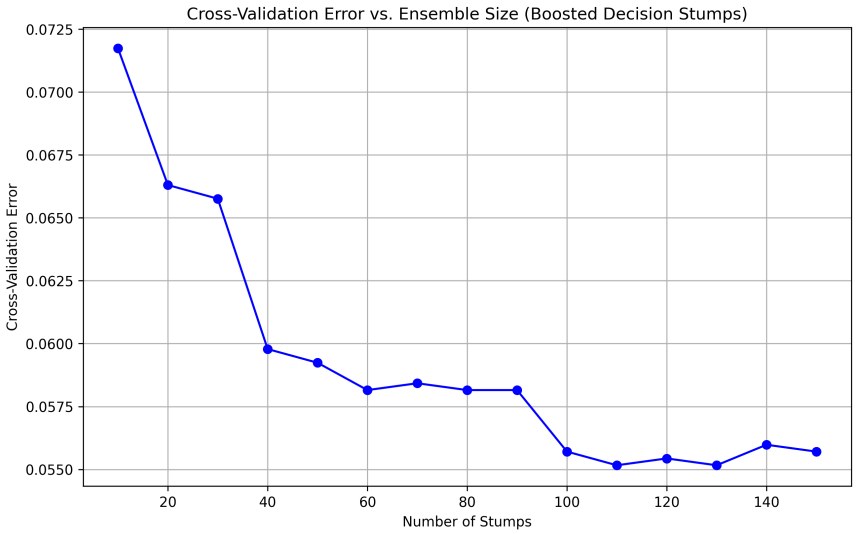

Figure 1: Cross-validation error as a function of ensemble size (number of stumps).

Key Findings The analysis reveals several crucial insights about AdaBoost behavior:

1. **Initial Improvement**: Substantial error reduction occurs in the first 40 stumps, highlighting the importance of minimum ensemble size.
2. **Optimization Point**: The model achieves optimal performance at 110 stumps, suggesting this as an ideal configuration point.
3. **Fold Sensitivity**: Higher variance between folds compared to random forests indicates greater sensitivity to data partitioning.
4. **Efficiency Threshold**: Minimal improvements beyond 110 stumps suggest this as an efficient cutoff point for ensemble size.

Conclusion The cross-validation analysis establishes 110 stumps as the optimal ensemble size, achieving a 5.52% cross-validation error while maintaining computational efficiency. This configuration represents an effective balance between model complexity and generalization ability. The higher variance between folds, compared to random forests, suggests that AdaBoost's performance is more dependent on the specific data split, though it maintains robust overall performance.

The analysis demonstrates that AdaBoost can achieve strong performance with moderate ensemble sizes, though with slightly higher sensitivity to data partitioning than random forests. This provides valuable insights for practical implementation, suggesting that while larger ensembles offer minimal benefits, careful consideration of data splitting may be more critical for optimal performance.

2.3 - Comparative Analysis

The comparative analysis of Random Forest and AdaBoost implementations reveals distinct performance characteristics and provides clear guidance for model selection on this dataset.

Performance Comparison Both models achieve their optimal performance with 110 estimators, though with markedly different error rates. The Random Forest achieves a cross-validation error of 4.40% and test error of 6.41%, while AdaBoost shows higher error rates with 5.52% cross-validation

error and 8.15% test error. Starting with just 10 estimators, Random Forest begins with a lower initial error of 5.27% compared to AdaBoost's 7.17%. After the first phase of improvement at 40 estimators, Random Forest maintains its advantage with 4.62% error versus AdaBoost's 5.98%.

Phase	Random Forest	AdaBoost
Initial Error (10 estimators)	5.27%	7.17%
Sharp Improvement Until	40 trees (4.62%)	40 stumps (5.98%)
Final Best CV Error	4.40%	5.52%
Test Error	6.41%	8.15%

Table 1: Comparison of Error Rates between Random Forest and AdaBoost

Model Characteristics The Random Forest demonstrates superior performance across all metrics, showing more consistent performance across different folds and better generalization with a smaller gap between cross-validation and test error. Its lower error rates at every stage of ensemble size scaling indicate a more robust learning approach. While AdaBoost achieves respectable performance, it exhibits higher variance between folds and a larger gap between cross-validation and test error, suggesting less stable generalization characteristics.

Error Reduction Patterns Both models display similar patterns in their error reduction trajectories. They show sharp improvement in the first 40 estimators, followed by a more gradual improvement phase until reaching 110 estimators. Beyond this point, additional estimators provide minimal performance gains, indicating a clear point of diminishing returns. This pattern holds true for both algorithms, though Random Forest maintains its performance advantage throughout the entire range of ensemble sizes.

Conclusion The analysis provides compelling evidence that Random Forest is the optimal choice for this dataset. Its superior accuracy is demonstrated by the 4.40% cross-validation error and 6.41% test error, significantly outperforming AdaBoost's corresponding metrics. The Random Forest also exhibits more robust generalization capabilities and more stable performance across different data splits, making it a more reliable predictor overall. The optimal configuration of 110 estimators represents an effective balance between model complexity and performance for both algorithms, though Random Forest maintains its advantage throughout all ensemble sizes. This comprehensive evaluation suggests that Random Forest should be the preferred model for this particular classification task, offering better accuracy, stability, and generalization capabilities.

3. Conclusion

This analysis explored the performance of three classification models - decision trees, random forests, and AdaBoost - on the Spambase Augmented dataset. Each model demonstrated distinct characteristics and trade-offs in handling this high-dimensional spam classification task.

Decision Trees showed clear signs of overfitting, achieving perfect training accuracy but test errors around 7.5-8%. While pruning and depth limitations improved generalization, the persistent gap between training and test performance highlights the model's fundamental limitations with high-dimensional data.

Random Forests demonstrated superior and more stable performance, achieving test errors of 6.5% and cross-validation errors as low as 4.4%. The ensemble approach effectively mitigated overfitting while maintaining strong generalization ability. The model showed remarkable stability across different feature subset sizes and achieved optimal performance with relatively few trees (40-110), suggesting efficient computational requirements.

AdaBoost with decision stumps achieved intermediate performance, with cross-validation errors of 5.52%. While showing higher variance between folds than random forests, it demonstrated an interesting property of increasing training error with more data, suggesting robust generalization. The model reached optimal performance with 110 stumps, similar to random forests.

Cross-validation analysis revealed that both ensemble methods benefit from larger ensemble sizes up to about 110 trees/stumps, after which returns diminish significantly. Random forests showed more consistent performance across folds ($\pm 0.5\%$ variation) compared to AdaBoost ($\pm 1\%$ variation), suggesting greater stability in handling different data partitions.

For this spam classification task, random forests emerge as the most effective approach, combining strong performance (4.4% error) with stable behavior across different configurations. The analysis demonstrates the value of ensemble methods in handling high-dimensional data while highlighting the importance of proper model selection and configuration for optimal performance.