Ensemble Learning

3945 – Advanced Machine Learning, Sem A 2024

Timor Baruch, Hadar Pur

Feb 5, 2024

1 AdaBoost implementation

1.1 Description of Implementation

AdaBoost, short for Adaptive Boosting, is an ensemble learning technique that combines the strengths of multiple weak learners to create a robust and accurate predictive model.

We implemented AdaBoost using a custom class named OurAdaBoost. The implementation follows the pseudocode taught in class, which is attached below as Algorithm 1. The class is initialized with parameters such as the number of estimators, learning rate, and random state. The fit method trains the AdaBoost model by iteratively selecting weak classifiers based on weighted errors and updating the weights. In our implementation, the weak classifiers used are decision trees with a depth of one, commonly referred to as stumps. The default learning rate is set to 1.

The predict method combines predictions from all weak classifiers, providing the final prediction for a given input. The score method evaluates the accuracy of the model by comparing its predictions against the true labels.

Algorithm 1 AdaBoost in Pseudo-Code

$H_0 \leftarrow 0$
$\forall_i : w_i \leftarrow \frac{1}{n}$
for $t = 0$ to $T - 1$ **do**
    $h \leftarrow \arg\min_h \sum_{i:h(x_i)\neq y_i} w_i$
    $\epsilon \leftarrow \sum_{i:h(x_i)\neq y_i} w_i$
    if $\epsilon < \frac{1}{2}$ **then**
        $\alpha \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon}{\epsilon}\right)$
        $H_{t+1} \leftarrow H_t + ah$
        $\forall_i : w_i \leftarrow w_i \frac{w_i e^{-ah(x_i)y_i}}{2\sqrt{\epsilon(1-\epsilon)}}$
    else
        return (H_t)
    end if
end for
return (H_T)

This implementation aims to leverage the power of AdaBoost to create an ensemble of decision stumps, enhancing the overall model's predictive capabilities. The modular structure of the code allows for easy customization of parameters and serves as a flexible tool for various classification tasks.

1.2 Data Generated for Tests

To evaluate the performance of our model against scikit-learn, we aimed to challenge the algorithm by comparing it with more complex datasets than classic toy ones. These datasets, while maintaining a relatively small number of samples and two features for simplicity, were deliberately designed to be more challenging than straightforward linearly separated ones. In order to examine our implementation in diverse scenarios, we generated two datasets:

- **First Dataset:** a synthetic dataset designed for testing with 10,000 samples, 2 features and binary labels (-1 or 1).
- **Second Dataset:** comprised of two binary-labeled datasets, with labels -1 or 1, each containing 2,000 samples with 2 features. Concatenation results in a larger dataset of 4,000 samples. This dataset was specifically designed to assess the model's performance on concatenated and diverse data.

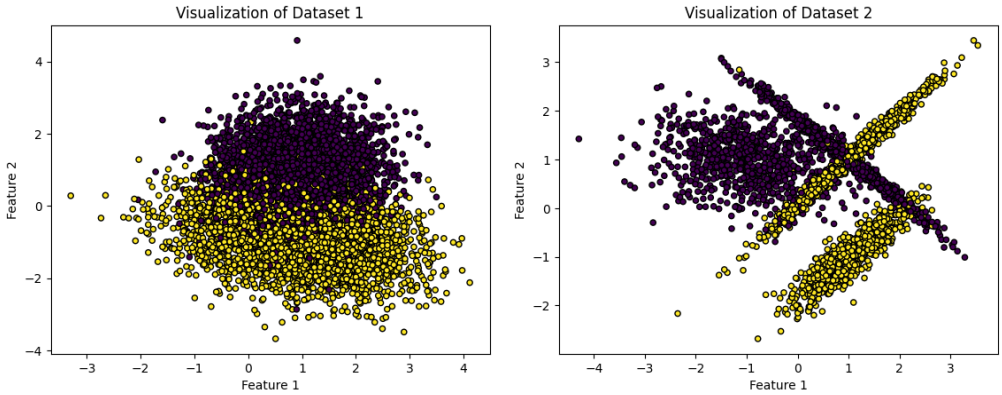


Figure 1: Datasets Visualizations

The datasets were uniformly split into training and test sets, maintaining a consistent 80%-20% ratio.

1.3 Tests and Results

Our AdaBoost implementation was applied to both datasets and compared against the official scikit-learn AdaBoost (AdaBoostClassifier). We recorded accuracy scores and other relevant metrics. The results demonstrate the effectiveness of our implementation on diverse datasets. To ensure a fair comparison, we used default settings, such as 50 estimators similar to scikit-learn's AdaBoost, and a default learning rate of 1. This approach facilitated a meaningful and consistent comparison between the two implementations.

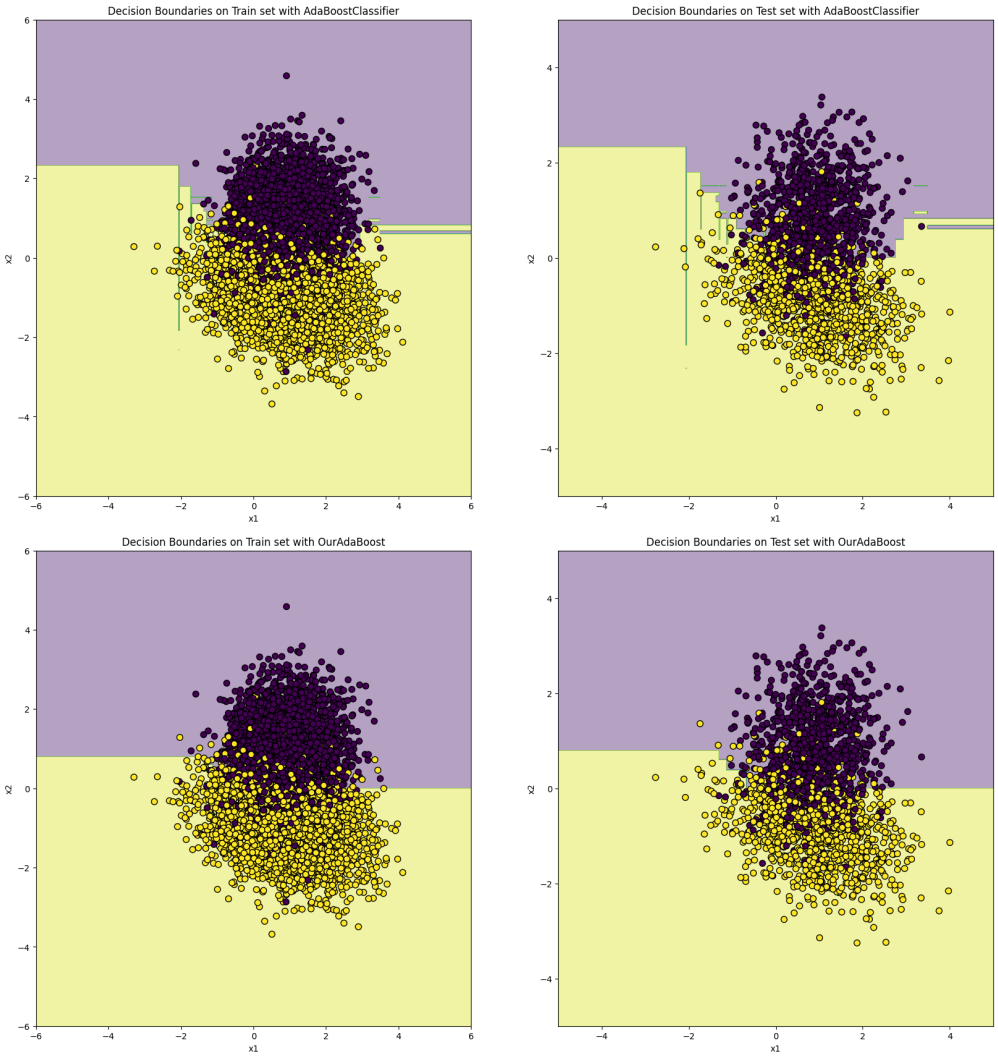


Figure 2: First dataset: Decision Boundaries Comparison between Our Implementation and Official AdaBoost

The decision boundaries in Figure 2 illustrate the classification behavior of the two AdaBoost implementations on the first dataset. The top plot shows the decision boundaries from the official AdaBoost implementation, while the bottom plot displays the decision boundaries from our AdaBoost implementation.

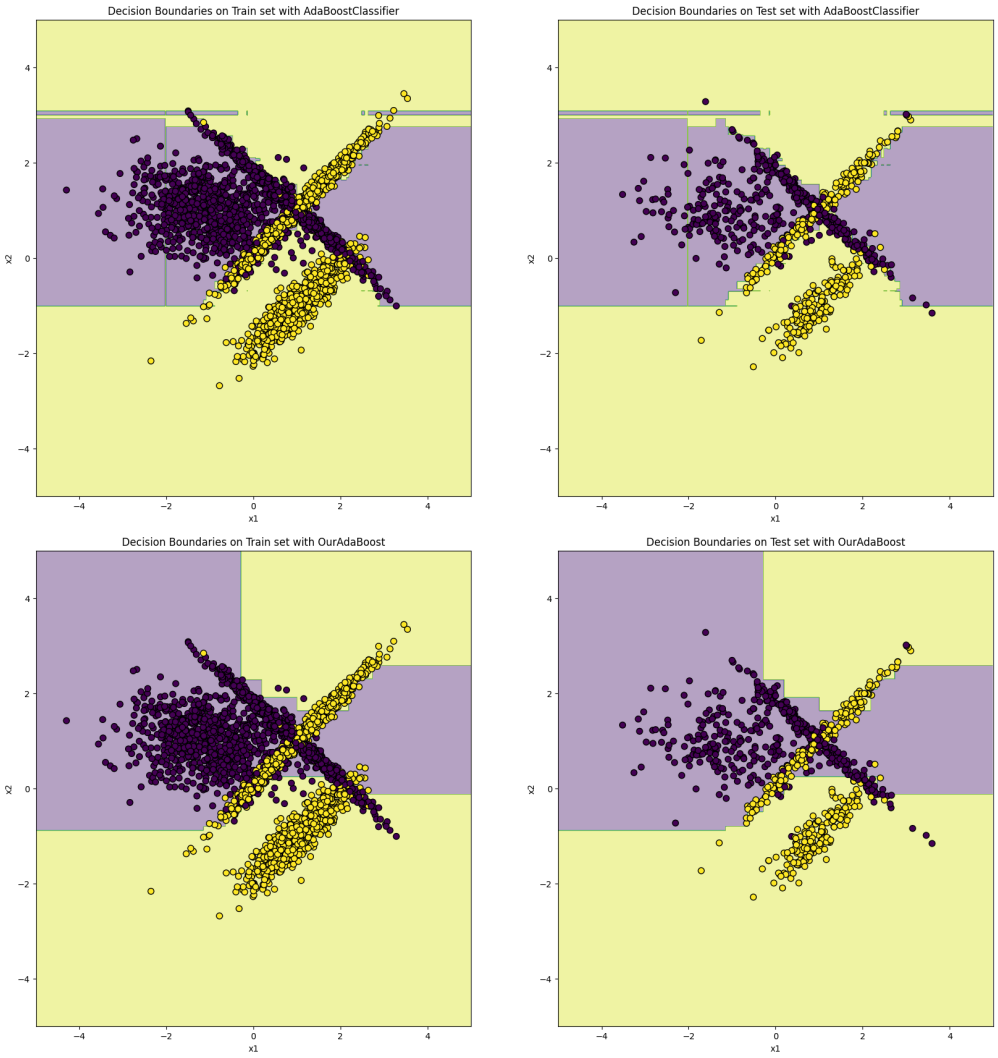


Figure 3: Second dataset: Decision Boundaries Comparison between Our Implementation and Official AdaBoost.

Similarly, Figure 3 presents the decision boundaries comparison for the second dataset. The top plot illustrates the decision boundaries generated by the official AdaBoost implementation, while the bottom plot depicts the decision boundaries produced by our AdaBoost implementation.

	Dataset	Our AdaBoost	scikit-learn AdaBoost
Train Set Accuracy	**1**	87.76%	88.16%
Test Set Accuracy	**1**	86.35%	86.85%
Train Set Accuracy	**2**	78.34%	81.47%
Test Set Accuracy	**2**	75.88%	74.12%

Table 1: Comparison of Train and Test Set Accuracy between Our AdaBoost and scikit-learn AdaBoost

As shown in Table 1, the comparison between Our AdaBoost and scikit-learn's AdaBoost on two datasets revealed similar overall performance. For the first dataset, both implementations demonstrated comparable train and test set accuracies. However, in the second dataset, Our AdaBoost showed a slightly lower train set accuracy but a slightly higher test set accuracy compared to scikit-learn's AdaBoost, maybe indicating that the official implementation started to overfit to the training data. Despite these slight variations, both implementations exhibited competitive results across the evaluated metrics.

2 Experiments

2.1 Generated Datasets

For this part, we created two distinct datasets with binary labels {-1, 1} to examine different aspects of our AdaBoost implementation:

- **First dataset:** involves non-linear separation and introduces some noise to challenge the algorithm's adaptability. It consists of 2000 samples with 10 features.
- **Second dataset:** intentionally imbalanced classes, where 70% of the samples belong to class -1 and 30% to class 1, featuring noise while linear separation. It consists of 2000 samples with 10 features. This dataset aims to evaluate how well Our AdaBoost handles imbalanced data and linearly separable cases.

  Both datasets were split into training and testing sets with 80%-20% ratio respectively, for comprehensive evaluation.

In order to improve the visualization of our results and decision boundaries, we used the PCA dimensionality reduction technique, to reduce the dimensionality of the datasets to two features.

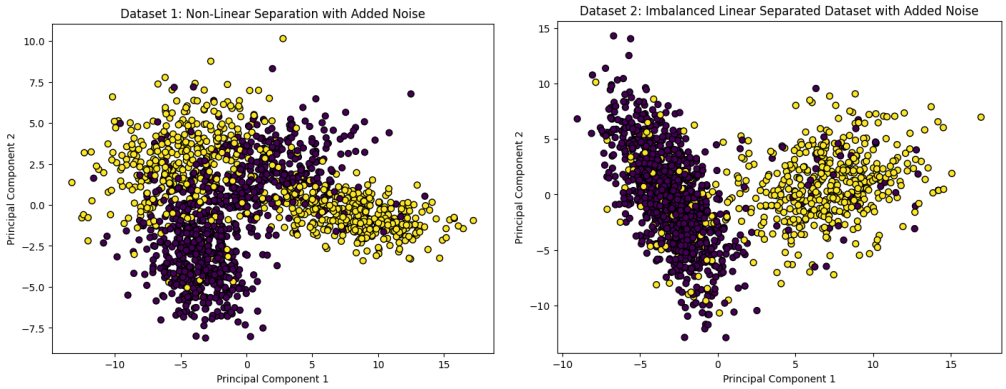


Figure 4: Datasets Visualizations After PCA

2.2 Hyperparameter Tuning

2.2.1 Hyperparameter Tuning for the First Dataset

To enhance the performance of our AdaBoost model on the first dataset, we conducted a thorough hyperparameter tuning process. This involved a grid search over different combinations of hyperparameters, specifically `n_estimators` (number of weak learners) and `learning_rate`.

The model was trained with each combination, and the accuracy, precision, recall, and F1 score were evaluated on the test set. These metrics offer a comprehensive assessment of the model's performance, considering overall correctness (accuracy), accuracy of positive predictions (precision), the ability to capture all positive instances (recall), and a balanced combination of precision and recall (F1 score). The set of hyperparameters that resulted in the highest accuracy for the first dataset was then selected for further analysis. The results are shown below.

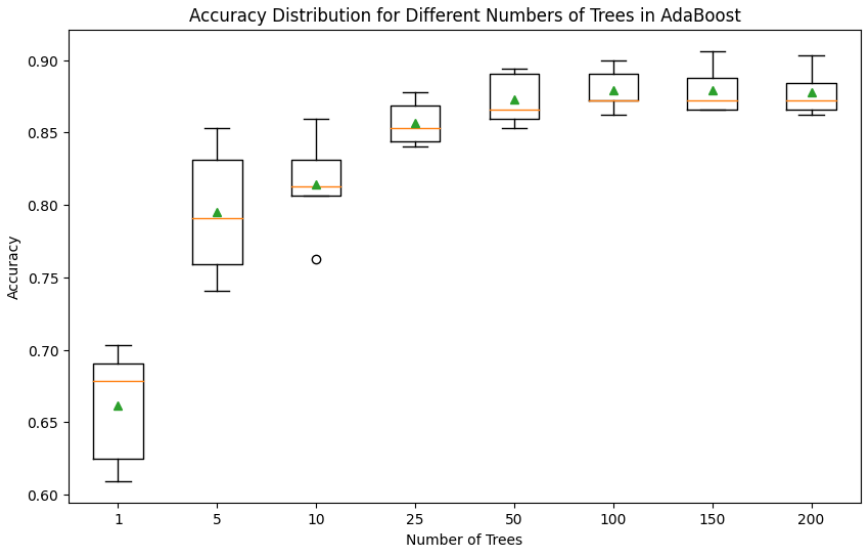


Figure 5: Accuracy vs Number of Trees for First Dataset

The table below provides an example of hyperparameter tuning test results for the first dataset, with the best combination highlighted in bold.

Estimators	Learning Rate	Accuracy	Precision	Recall	F1 Score
1	0.001	0.69	0.64	0.93	0.76
5	0.5	0.83	0.83	0.84	0.84
10	1.0	0.84	0.83	0.86	0.84
50	1.0	0.87	0.84	0.92	0.88
100	0.9	0.87	0.84	0.91	0.87
200	**1.0**	**0.89**	**0.87**	**0.92**	**0.89**

Table 2: Hyperparameter Combinations - Dataset 1

2.2.2 Hyperparameter Tuning for the Second Dataset

In a similar fashion, we carried out a hyperparameter tuning process for the second dataset. Employing a grid search with variations in `n_estimators` and `learning_rate`, we trained the AdaBoost model on different combinations and assessed its performance metrics as for the first dataset. The hyperparameters that yielded the highest accuracy on the second dataset were identified as the optimal configuration for subsequent analysis. The results are shown below.

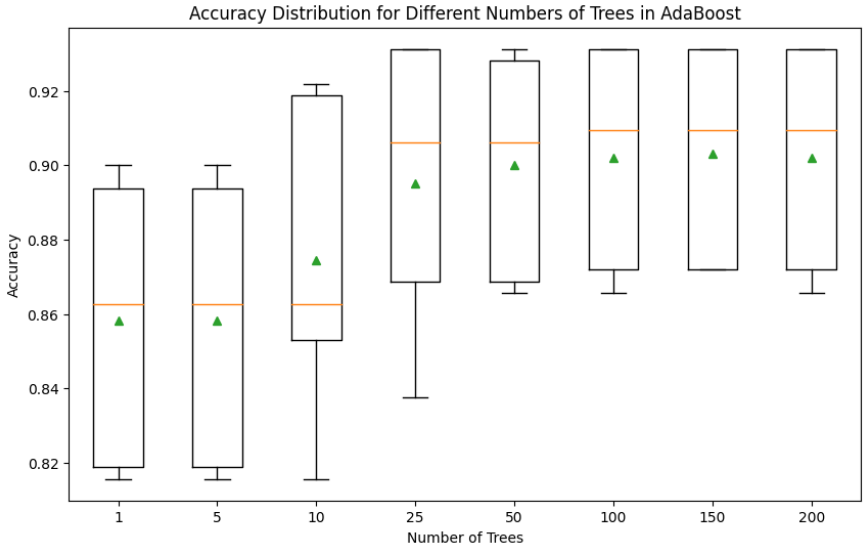


Figure 6: Accuracy vs Number of Trees for Second Dataset

The table below provides an example of hyperparameter tuning test results for second dataset, with the best combination highlighted in bold.

Estimators	Learning Rate	Accuracy	Precision	Recall	F1 Score
1	0.01	0.87	0.85	0.75	0.80
5	0.1	0.89	0.88	0.75	0.81
10	0.5	0.91	0.91	0.80	0.85
25	0.9	0.91	0.92	0.80	0.85
50	**1.0**	**0.92**	**0.92**	**0.82**	**0.87**
200	0.9	0.91	0.92	0.81	0.86

Table 3: Hyperparameter Combinations - Dataset 2

2.2.3 Summary of test results

As observed in figures 5 and 6, there is a pattern in the influence of the number of trees on accuracy. For configurations with a single-digit number of trees, the accuracy is notably lower. However, starting from around 25 trees and beyond, the accuracy stabilizes and remains relatively constant. This trend suggests that the default setting of approximately 50 trees is effective, demonstrating consistent performance across various datasets. It is crucial to be mindful of this behavior as it implies that using a larger number of trees may not necessarily lead to substantial accuracy improvements and could potentially introduce overfitting by capturing noise or specific details in the training data.

For the first dataset evaluation as shown in table 2, the optimal parameters are identified as 200 estimators with a default learning rate of 1. However, the difference in accuracy and metric scores between these best hyperparameters and the default setting of 50 estimators is minor, with an improvement from 87% to 89% in accuracy while the same recall score. This suggests that the default parameters, particularly with 50 estimators, perform reasonably well.

However, as observed in table 3 for the second dataset evaluation, the default parameters are found to be optimal. This observation indicates that AdaBoost leverages robust default parameters that are well-suited for achieving optimal performance across diverse datasets.

2.3 Comparisons with other Classification Models

In this section, we implemented and compared the performance of our custom AdaBoost classifier ('OurAdaBoost') with three other classification models: 'AdaBoostClassifier' from the scikit-learn library, a Linear Support Vector Machine (SVM), and a Random Forest classifier. For each classification model we used for comparison we used default implementation.

We assessed the classification models using metrics such as accuracy, precision, recall, and F1 score on the test datasets after training each model on corresponding training datasets. These metrics offer insights into the models' classification abilities. The next step involves creating plots to visualize performance differences across models, providing a comprehensive understanding of how our custom AdaBoost model compares with other classification algorithms.

We used for visualizations the PCA (2 components) dimentionality reduction technique as demonstrated in Figures 8, 11. The results are shown in the following parts.

2.3.1 Comparisons results for First Dataset

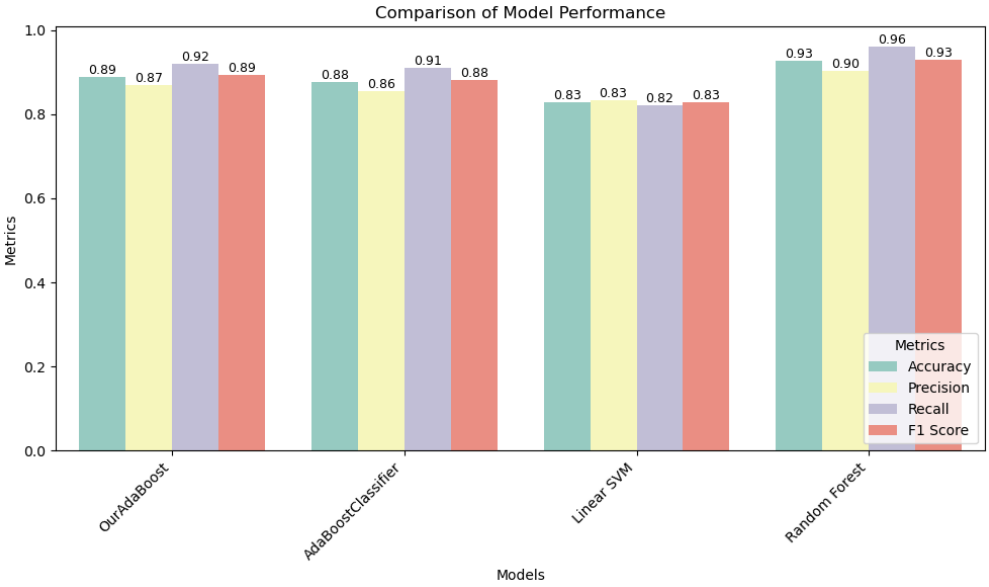
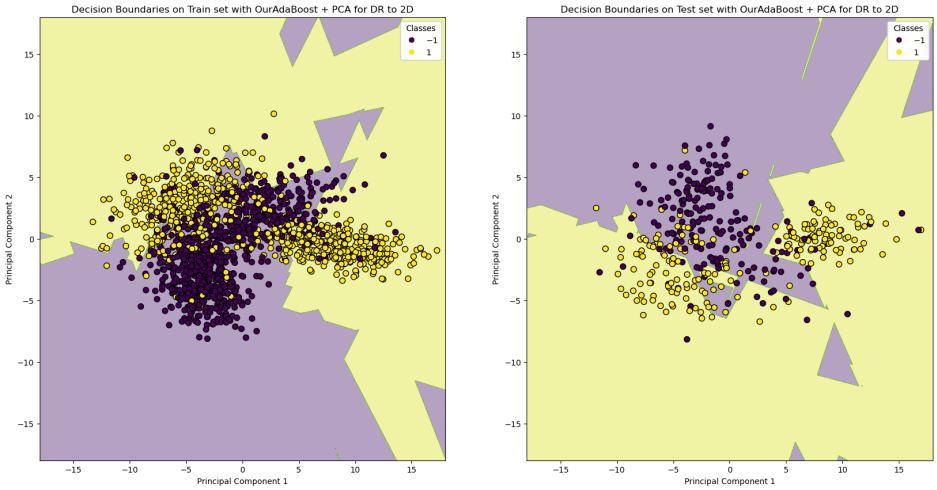


Figure 7: Performance Metrics Comparison Across Classification Models

2.3.2 Decision Boundaries for First Dataset



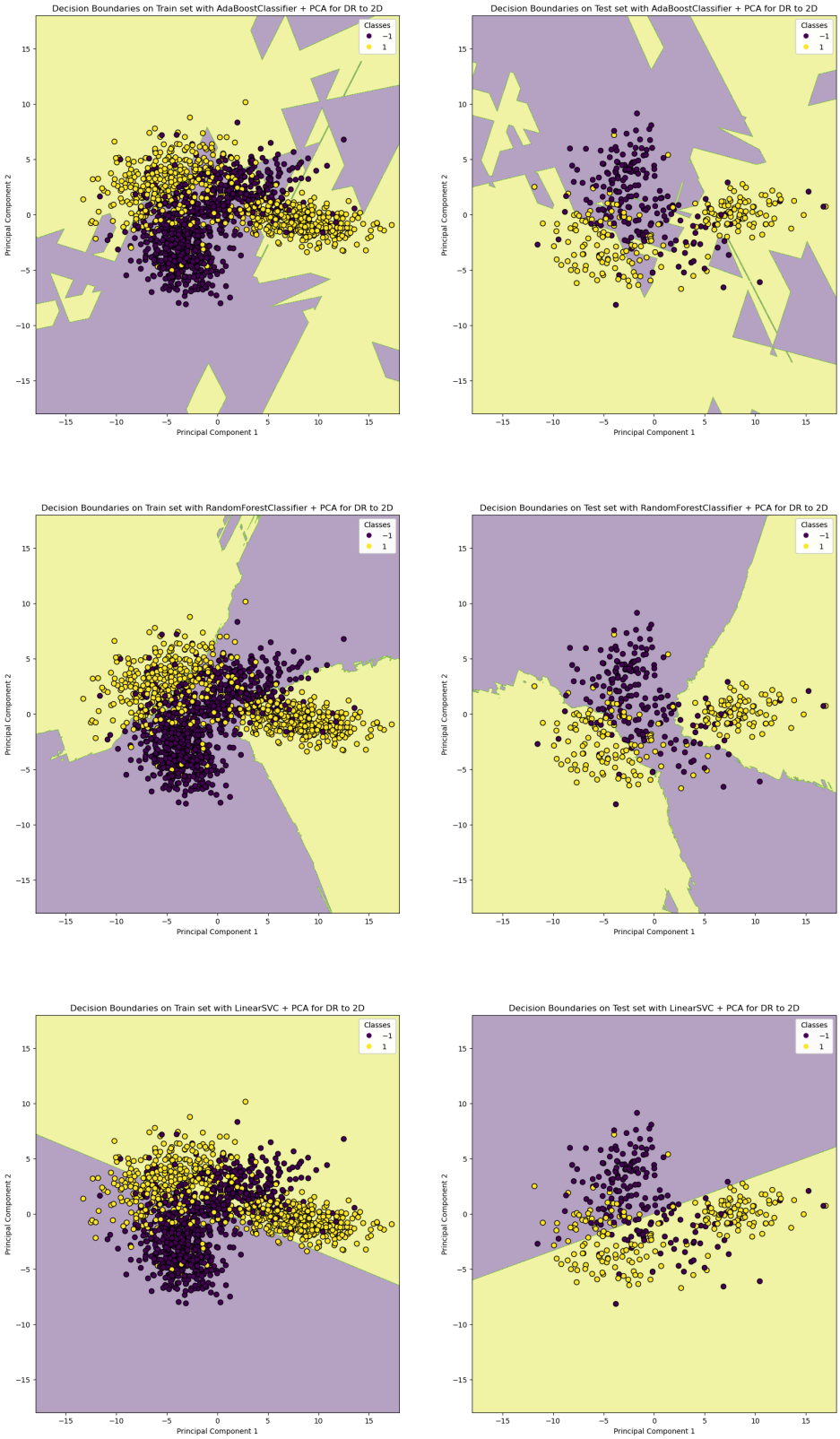


Figure 8: Decision boundaries Comparison Across Classification Models

2.3.3 Confusion Matrices for First Dataset

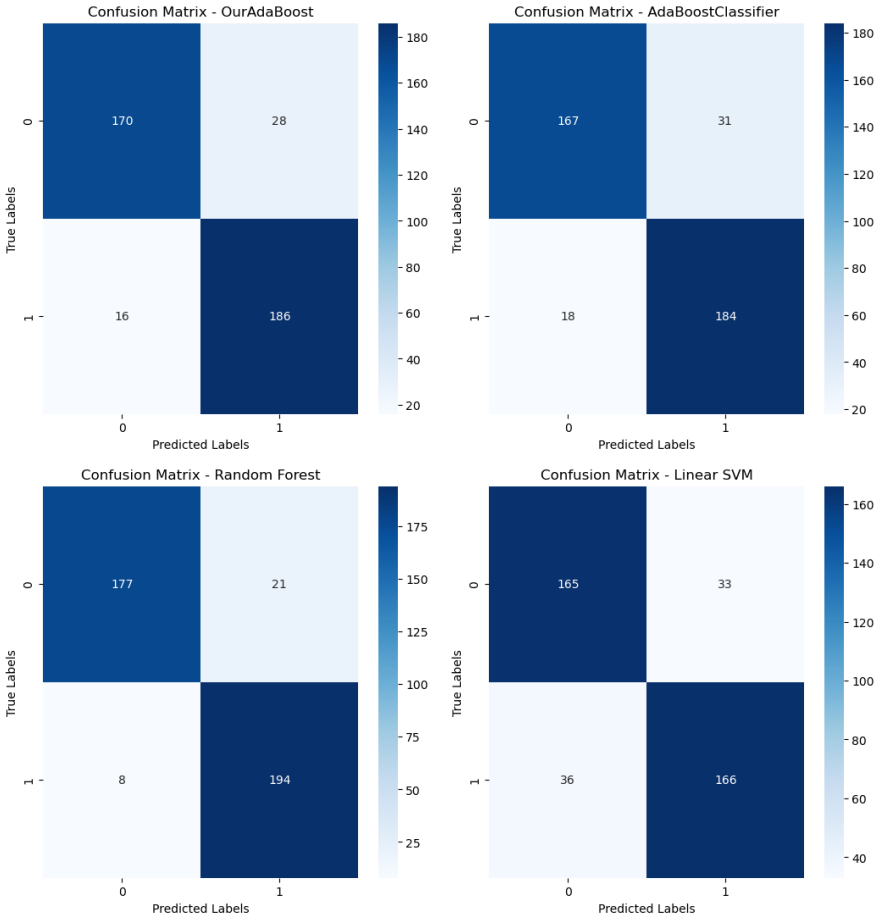


Figure 9: Confusion Matrices Comparison Across Classification Models

The results highlight notable variations in classification model performance. Random Forest excelled with a perfect training accuracy of 100% and a strong 92.75% test accuracy, showcasing its ability to capture intricate data patterns. In contrast, Support Vector Machine (SVM) displayed comparatively lower performance (82.25% training accuracy, 82.75% test accuracy), possibly due to the non-linearity of the dataset.

Our custom AdaBoost model demonstrated competitive performance (90.38% training accuracy, 89.00% test accuracy), showcasing its robust generalization capabilities. These findings underscore the importance of selecting models aligned with dataset characteristics, with Random Forest proving versatile in handling complex relationships.

2.3.4 Comparisons results for Second Dataset

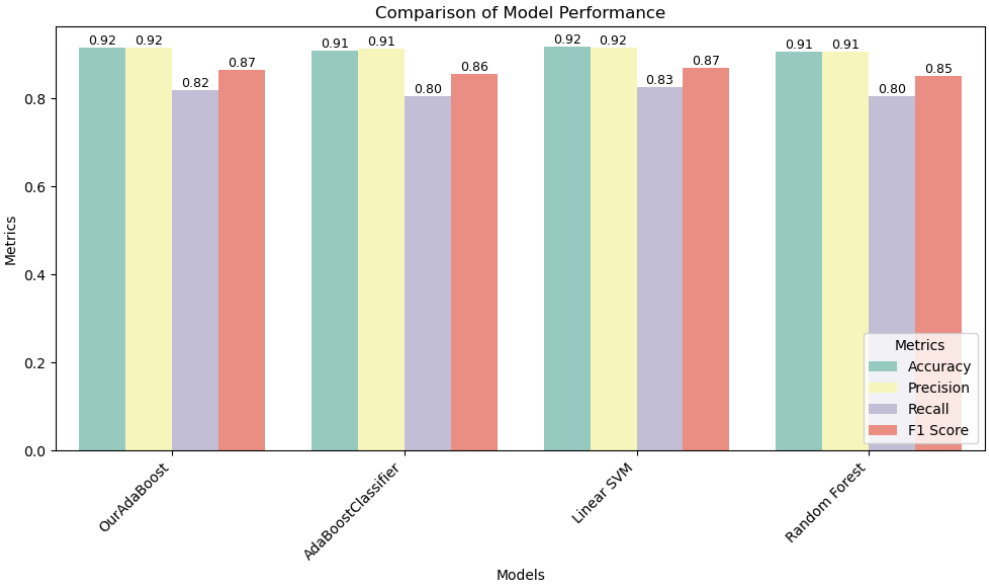
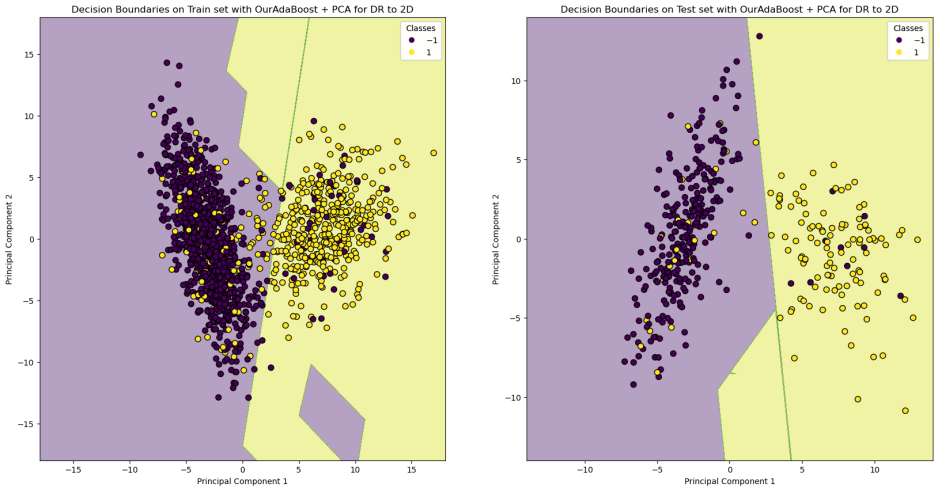


Figure 10: Performance Metrics Comparison Across Classification Models

2.3.5 Decision Boundaries for Second Dataset



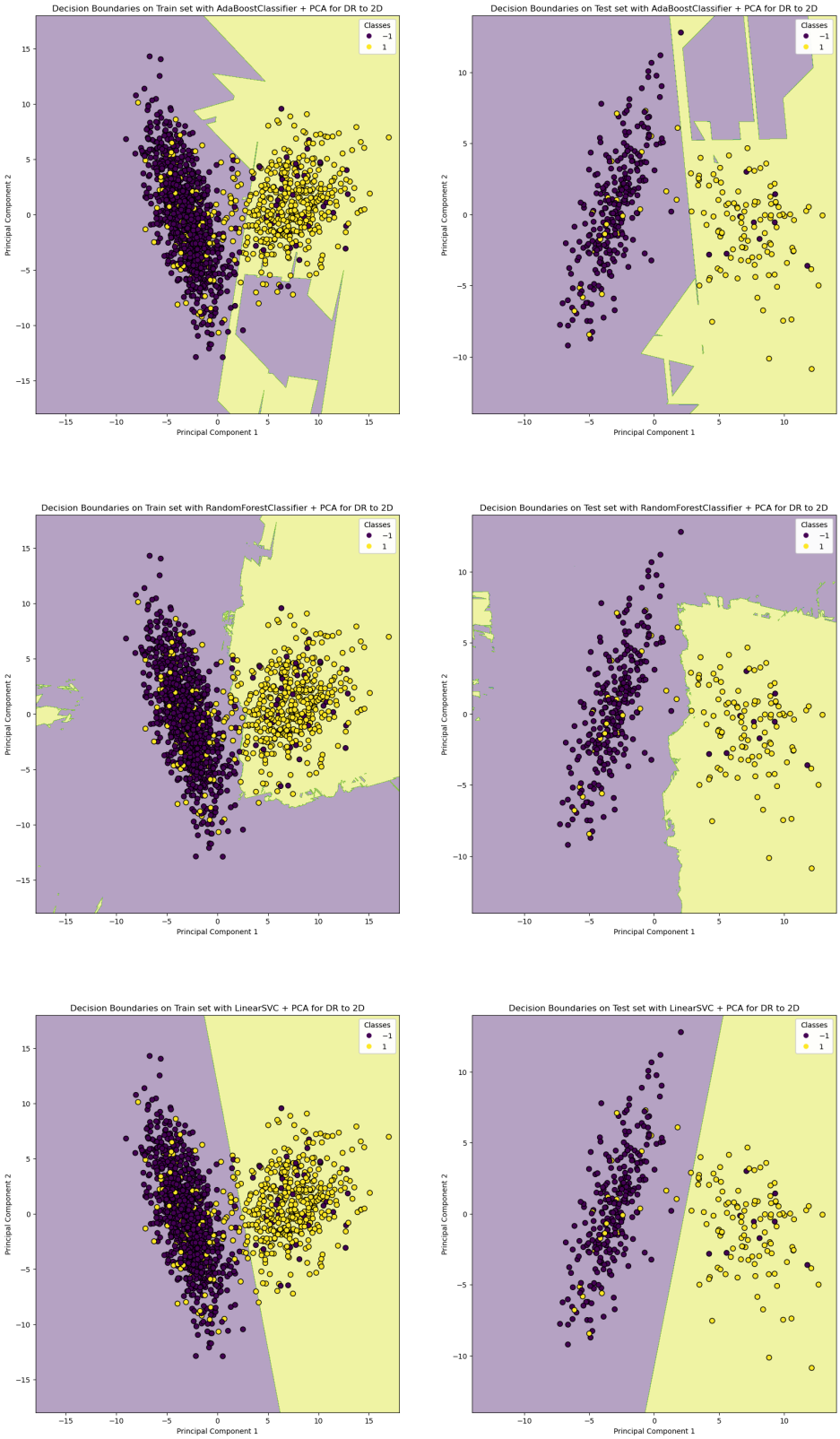


Figure 11: Decision boundaries Comparison Across Classification Models

2.3.6 Confusion Matrices for Second Dataset

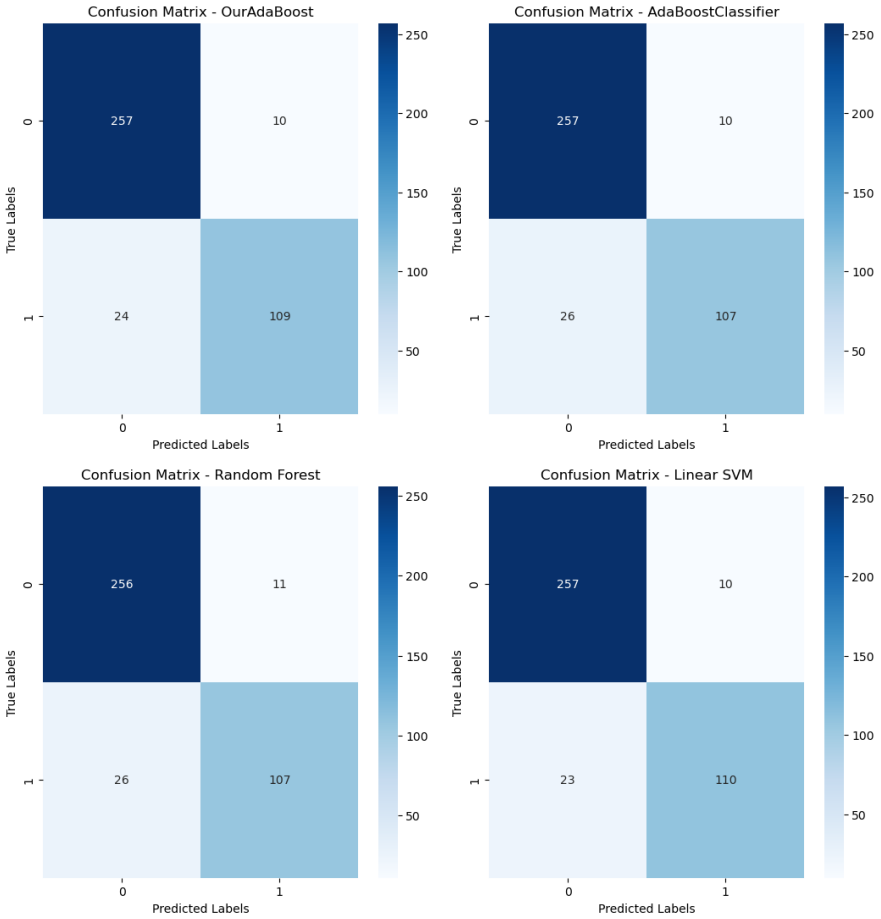


Figure 12: Confusion Matrices Comparison Across Classification Models

The results indicate that our custom AdaBoost model performed well on this imbalanced dataset, achieving a train accuracy of 90.75% and a test accuracy of 91.50%. The original AdaBoost model also demonstrated strong performance with a train accuracy of 90.88% and a test accuracy of 91.00%. Support Vector Machine (SVM) showcased high accuracy (91.06% train, 91.75% test), benefiting from the dataset's linear separability. However, Random Forest (RF) exhibited overfitting (100.00% train, 90.75% test), likely influenced by the imbalanced class distribution and noise in the data. The imbalance may have contributed to RF's sensitivity to the majority class, while our AdaBoost model's robustness against overfitting, even in the presence of noise, underscores its suitability for challenging datasets.

3 Summary and discussion

In Conclusion, our work focused on developing and evaluating a custom AdaBoost model, comparing it with established classification algorithms, namely AdaBoost, Support Vector Machine (SVM), and Random Forest (RF).

In the initial phase, we generated two toy binary datasets— one composed of concatenated 4,000 samples from 2 different datasets and the other containing 10,000 linearly separable samples. Both datasets were simplified to only two features for a straightforward comparison with official AdaBoost results, revealing notable similarities in the test accuracy score and decision boundaries.

In the subsequent stage, we designed two more complex binary datasets; one with 10 features showcasing nonlinear separation and the other with 10 features demonstrating linear separation but imbalanced classes, both with added noise. The aim was to illustrate AdaBoost's advantages, highlighting its versatility, robustness against overfitting, simplicity of model parameters, and effective handling of imbalanced datasets.

We conducted an exhaustive hyperparameter tuning process through grid search, identifying optimal parameters for our AdaBoost model on each dataset. Evaluations and comparisons between the models involved key metrics such as accuracy, precision, recall, and F1 score. The custom AdaBoost model demonstrated competitive performance, adeptly avoiding overfitting and highlighting its resilience against noise and imbalanced class distributions.

In summary, our work focused on developing and evaluating a custom AdaBoost model, showcasing its advantages over established classifiers like official AdaBoost, SVM, and RF. Through extensive hyperparameter tuning and evaluations on diverse datasets, our AdaBoost model demonstrated competitive performance, emphasizing its versatility and robustness.