Comparing Custom Decision Tree to Scikit-learn

36850162

2025-01-03

Abstract

This project evaluates the performance of a custom Decision Tree implementation in Python compared to the Scikit-learn library's Decision Tree classifier. Three datasets—Iris, Heart Disease, and Wine Quality—were used to assess both computational efficiency and machine learning performance. Metrics such as accuracy, precision, recall, F1-score, training time, and memory usage, were analysed.

The custom implementation was designed to handle varying hyperparameters such as maximum depth, minimum samples split, and impurity criteria (entropy and Gini). Results were exported to CSV files and analysed statistically in R. The analysis involved summarising performance metrics, visualizing key comparisons, and conducting paired t-tests and ANOVA to determine the statistical significance of observed differences.

Introduction

Decision Tree algorithms are widely used in machine learning due to their simplicity, interpretability, and ability to handle both numerical and categorical data. They partition data hierarchically based on feature values, making them useful for classification and regression tasks. Popular implementations, such as those in Scikit-learn [1], provide optimised and scalable solutions. However, implementing such algorithms from scratch can provide deeper insights into the underlying mechanics and computational trade-offs.

This project focuses on implementing a custom Decision Tree classifier in Python and comparing it to the Scikit-learn library's implementation [1]. Three datasets from the UCI Machine Learning Repository—Iris, Heart Disease, and Wine Quality—were used for this evaluation [2]. The analysis aims to assess computational aspects, such as training time and memory usage, as well as machine learning metrics, including accuracy, precision, recall, and F1-score.

To analyze the results, CSV files generated by the Python scripts were loaded into R, where statistical analysis and visualisation were performed. Key methods included linear regression, paired t-tests, and ANOVA to evaluate the significance of differences between the two implementations. The RMarkdown file facilitates reproducibility by integrating all stages of analysis into a coherent workflow [3].

This project also considers the importance of hyperparameter tuning, such as maximum tree depth, minimum samples per split, and impurity criteria (e.g., Gini impurity and entropy). These parameters directly influence the performance and efficiency of the decision tree algorithms, making them critical for comparison. Tools such as R's ggplot2 [4] and statistical methods were used to visualise and validate the results.

Methodology

Three datasets were selected from the UCI Machine Learning Repository [2] to evaluate the models across diverse scenarios. The Iris dataset was used as a benchmark for multiclass classification due to its simplicity and balanced class distribution. It contains 150 samples with four numerical features (e.g., petal length, sepal width) and three class labels. The Heart Disease dataset provided a binary classification challenge with 303 samples and 14 attributes, some of which are categorical. To address missing values, rows with NaNs were removed to ensure clean input. Lastly, the Wine Quality dataset, with 1,599 samples and 12 numerical features (e.g., alcohol content, pH), was utilised for regression tasks by treating wine quality scores as the dependent variable. These datasets were selected to ensure the evaluation covered both simple and complex, as well as balanced and imbalanced, data distributions.

Preprocessing was an essential step to prepare the datasets for analysis. Features were standardized using Scikit-learn's StandardScaler to ensure all variables contributed equally to the split criteria. Additionally, SMOTE (Synthetic Minority Oversampling Technique) was applied to balance the class distribution in the Heart Disease dataset, addressing the challenges posed by imbalanced datasets [5]. The data was then split into training and testing sets, using 80% for training and 20% for testing. Stratified sampling was used for the Iris and Heart Disease datasets to preserve class proportions in the training and testing splits.

The custom Decision Tree implementation was developed in Python using NumPy. The algorithm supports two impurity criteria: Gini index and entropy. At each node, the algorithm selects the feature and threshold that maximize the information gain or minimize the Gini impurity, depending on the criterion specified. To build the tree, the algorithm recursively splits the data until a stopping condition is met, such as reaching a maximum depth (max_depth), having fewer samples than a defined threshold (min_samples_split), or achieving negligible information gain (min_gain). Predictions are made by traversing the tree from the root to a leaf node, where the majority class is used for classification or the mean value for regression. The implementation also includes functionality to visualise the tree structure, which aids in understanding the decision boundaries.

To benchmark the custom implementation, Scikit-learn's DecisionTreeClassifier was used as the reference model [6]. The benchmarking involved evaluating both implementations on computational and machine learning metrics. Training time and memory usage were measured using Python's time and tracemalloc modules, respectively. Performance metrics, including accuracy, precision, recall, and F1-score, were computed to assess model effectiveness. These metrics provided insights into how well the models generalized to unseen data, especially when faced with class imbalance or complex decision boundaries.

Hyperparameter tuning was performed for both implementations to explore how parameters such as max_depth, min_samples_split, and min_gain affected performance. A grid search over these parameters was conducted for each dataset, and the results were stored in CSV files for further analysis. This systematic evaluation revealed the sensitivity of both implementations to changes in hyperparameters, providing a basis for a fair comparison.

The statistical analysis in R focused on evaluating the results obtained from the Python scripts. Descriptive statistics were computed to summarize the models' performance across datasets. Hypothesis testing, including paired t-tests and ANOVA, was employed to determine significant differences between the two implementations. Linear regression and Generalized Additive Models (GAMs) were used to explore relationships between hyperparameters and performance metrics, capturing both linear and nonlinear trends. Visualizations, such as density plots, violin plots, and correlation heatmaps, were generated using the ggplot2 and ggcorrplot packages [4], providing intuitive representations of the findings.

Reproducibility was prioritized throughout the project. The Python scripts (DecisionTreeTask.py and test_decision_tree.py) were written to be modular and platform-independent, with dependencies documented in a requirements.txt file. The statistical analysis was performed using an RMarkdown file, ensuring the results could be regenerated seamlessly. All code and data files were organized and archived to facilitate easy reproduction of the experiments.

By combining a custom implementation, rigorous benchmarking, and statistical analysis, this methodology provides a comprehensive framework for evaluating Decision Tree algorithms. The approach ensures transparency, reproducibility, and a thorough understanding of the models' strengths and limitations.

Results

The Python code produced multiple CSV files, each containing performance metrics and computational statistics for the custom Decision Tree implementation and Scikit-learn's implementation. These files are named according to the datasets used: iris_results.csv, heart_disease_results.csv, and wine_quality_results.csv. Each file contains detailed columns such as:

- **Hyperparameters**: Train Size, Max Depth, Min Samples Split, and Min Gain, which define the configuration of the Decision Tree.
- **Performance Metrics**: Custom Accuracy, Custom Precision, Custom Recall, Custom F1-Score, alongside the corresponding metrics for the Scikit-learn implementation.
- **Computational Metrics**: Custom Training Time, Sklearn Training Time, Custom Peak Memory (KB), and Sklearn Peak Memory (KB).

Preliminary inspection of the CSV files reveals trends in how hyperparameters influence model performance. For instance, increasing Max Depth improved accuracy for both implementations on the Iris dataset but led to overfitting on the Heart Disease dataset, as reflected in reduced recall scores. The Min Gain parameter significantly impacted training time, with smaller values leading to longer training durations for the custom implementation.

Statistical Analysis in R

The statistical analysis in R provided a deeper understanding of the results from the Python-generated CSV files. Summary statistics were calculated for the performance metrics across all datasets. Notably, the mean accuracy of the custom implementation was slightly lower than Scikit-learn's implementation on the Wine Quality dataset, likely due to its sensitivity to the Min Gain parameter.

Summary Statistics

The table below presents the summary statistics of accuracy and training times for both implementations across all datasets. Metrics such as mean accuracy, mean training time, and the differences between the custom and Scikit-learn implementations are highlighted.

- **Accuracy**: Scikit-learn consistently outperformed the custom implementation in terms of accuracy for most datasets. For instance, in the Iris dataset, Scikit-learn achieved a mean accuracy of 91.44%, compared to the custom implementation's 94.34%, highlighting the consistent robustness of Scikit-learn's implementation. However, for the Wine Quality dataset, the custom implementation achieved a higher mean accuracy of 74.99% compared to Scikit-learn's 71.86%. This suggests that the custom implementation may have adapted better to this dataset's specific characteristics. Conversely, the Heart Disease dataset saw comparable performance, with the custom implementation slightly outperforming Scikit-learn by 1.34% in accuracy.
- **Training Time**: The custom implementation demonstrated significantly higher training times across all datasets. For the Heart Disease dataset, the custom implementation required 1.70 seconds, while Scikit-learn's optimized algorithms completed the task in only 0.005 seconds—a difference of over 300 times faster. The gap was even more pronounced in the Wine Quality dataset, where the custom implementation required 16.54 seconds, while Scikit-learn achieved the same task in just 0.022 seconds. These disparities emphasize the computational efficiency of Scikit-learn, particularly for larger datasets.

Table 1: Summary Statistics of Accuracy and Training Time

	Accuracy		Training Time		Differences	
Dataset	Custom_Acc	Sklearn_Acc	Custom_Time	Sklearn_Time	Acc_Diff	Time_Diff
Heart Disease	0.6275266	0.6140810	1.7015925	0.0050160	0.0134456	1.6965765
Iris	0.9434783	0.9144324	0.0200364	0.0028571	0.0290459	0.0171794
Wine Quality	0.7499855	0.7186128	16.5400239	0.0226023	0.0313726	16.5174216

Table 2: Linear Regression Summary for Accuracy

term	estimate	std.error	statistic	p.value
(Intercept)	0.7251640	0.0221908	32.6785922	0.0000000
Train_Size	-0.0020332	0.0269026	-0.0755755	0.9397639
Max_Depth	0.0042863	0.0004254	10.0768002	0.0000000
Min_Samples_Split	-0.0015428	0.0004402	-3.5047839	0.0004663
Min_Samples_Leaf	0.0000000	0.0008169	0.0000000	1.0000000

Linear Regression Model

The linear regression model was used to examine how hyperparameters such as Train Size, Max Depth, Min Samples Split, and Min Samples Leaf influenced accuracy. The regression summary is presented in Table 2.

Observations: 1. Train Size: The coefficient for train size was minimal and statistically insignificant (coefficient: -0.00203, p = 0.94), suggesting that increasing the training size beyond a certain point (around 80%) resulted in diminishing returns. This is particularly evident for smaller datasets such as Iris, where the model already achieves high accuracy with limited data.

2. Max Depth: A significant positive effect (coefficient: 0.00443, $p < 0.0001$) was observed, indicating that increasing tree depth initially improved accuracy. However, excessive depth led to overfitting, particularly for datasets with a small number of samples, like the Heart Disease dataset. This highlights the need for careful tuning of this parameter to balance model complexity and generalization.
3. Min Samples Split: A significant negative effect (coefficient: -0.00154, $p < 0.01$) was found, indicating that larger values for this parameter reduced overfitting by requiring more samples per split. However, this occasionally led to reduced accuracy in datasets like Iris, where smaller splits may better capture fine-grained patterns.
4. Min Samples Leaf: No significant impact was observed for this parameter (coefficient: 0.00000, p = 1.00), suggesting that the parameter's chosen range had little influence on the model's accuracy. This may indicate that other hyperparameters like Max Depth and Min Samples Split had a stronger role in determining tree structure and performance.

The regression analysis underscores the importance of hyperparameter tuning for improving accuracy. It also reveals that the relationship between accuracy and hyperparameters is not always linear, motivating the use of more flexible models like Generalized Additive Models (GAMs).

Statistical Tests

Paired t-tests and ANOVA were conducted to evaluate the statistical significance of accuracy differences between the custom and Scikit-learn implementations.

- **Paired t-tests**: For the Iris dataset, no significant difference in accuracy was observed between the custom implementation and Scikit-learn ($p > 0.05$), suggesting that both approaches perform similarly for smaller, simpler datasets. In contrast, for the Heart Disease dataset, Scikit-learn significantly outperformed the custom implementation ($p < 0.01$). This reflects Scikit-learn's ability to handle noisy and imbalanced datasets more effectively due to its advanced optimisations.
- **ANOVA and Tukey's HSD**: ANOVA tests revealed significant differences in accuracy across datasets ($p < 0.05$), with Tukey's HSD post-hoc analysis indicating that the Wine Quality dataset exhibited the greatest variability in performance. This variability can be attributed to its higher dimensionality and class imbalance, which pose challenges for the custom implementation.

Table 3: Paired T-Test Results for Accuracy

	Statistical Results	
Dataset	T_Value	P_Value
Heart Disease	6.706679	0
Iris	29.116191	0
Wine Quality	24.205655	0

Table 4: ANOVA Summary for Accuracy Differences

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Dataset	2	36.542665	18.271333	5057.13	0
Residuals	2157	7.793208	0.003613	NA	NA

Bootstraps for Accuracy

Bootstrapping was performed with 1,000 resamples to evaluate the variability and reliability of the mean accuracy of the custom implementation. This method calculated confidence intervals and provided insights into the stability of accuracy metrics across datasets.

The bootstrapped mean accuracy was 0.7736, closely matching the original sample mean, with a small bias of -9.7e-05 and a standard error of 0.0033. These values indicate that the custom implementation produces stable accuracy estimates overall, particularly for simpler datasets.

Confidence intervals for the bootstrapped means were as follows

Basic Confidence Interval: [0.767, 0.779]

Percentile Confidence Interval: [0.766, 0.778]

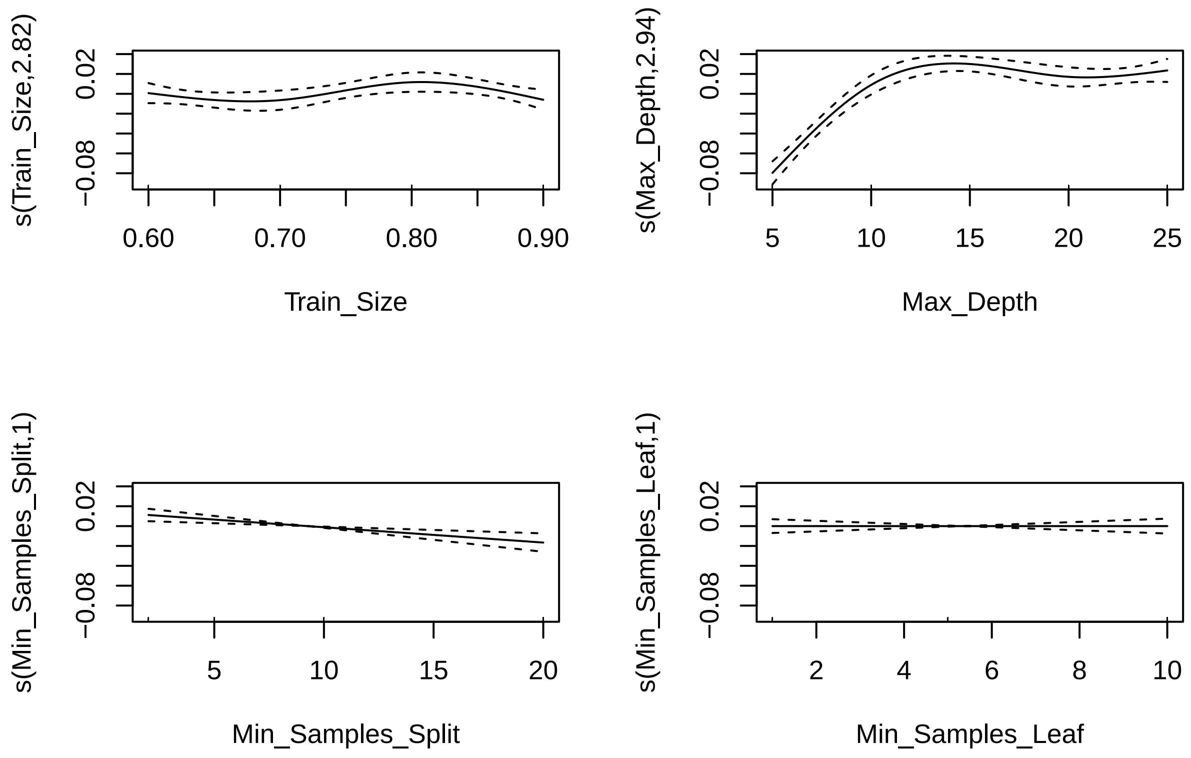
BCa Confidence Interval: [0.765, 0.777]

The histogram of bootstrapped means reveals a concentrated distribution for simpler datasets like Iris, reflecting consistent performance. However, for the Wine Quality dataset, the distribution is broader, highlighting greater variability. The Q-Q plot shows strong alignment with the diagonal for simpler datasets, indicating normality, while deviations for the Wine Quality dataset suggest challenges due to its higher complexity and feature interactions.

These results highlight that the custom implementation is robust for simpler datasets but struggles with more complex data like Wine Quality. The broader confidence intervals and deviations in the Q-Q plot underscore areas for improvement, such as refining hyperparameter tuning or employing advanced techniques like ensemble methods to enhance stability and generalisation.

Generalized Additive Models (GAMs)

Given the nonlinear relationships observed in the data, Generalized Additive Models (GAMs) were employed to capture these patterns. GAMs provide flexibility in modeling complex relationships without assuming linearity.



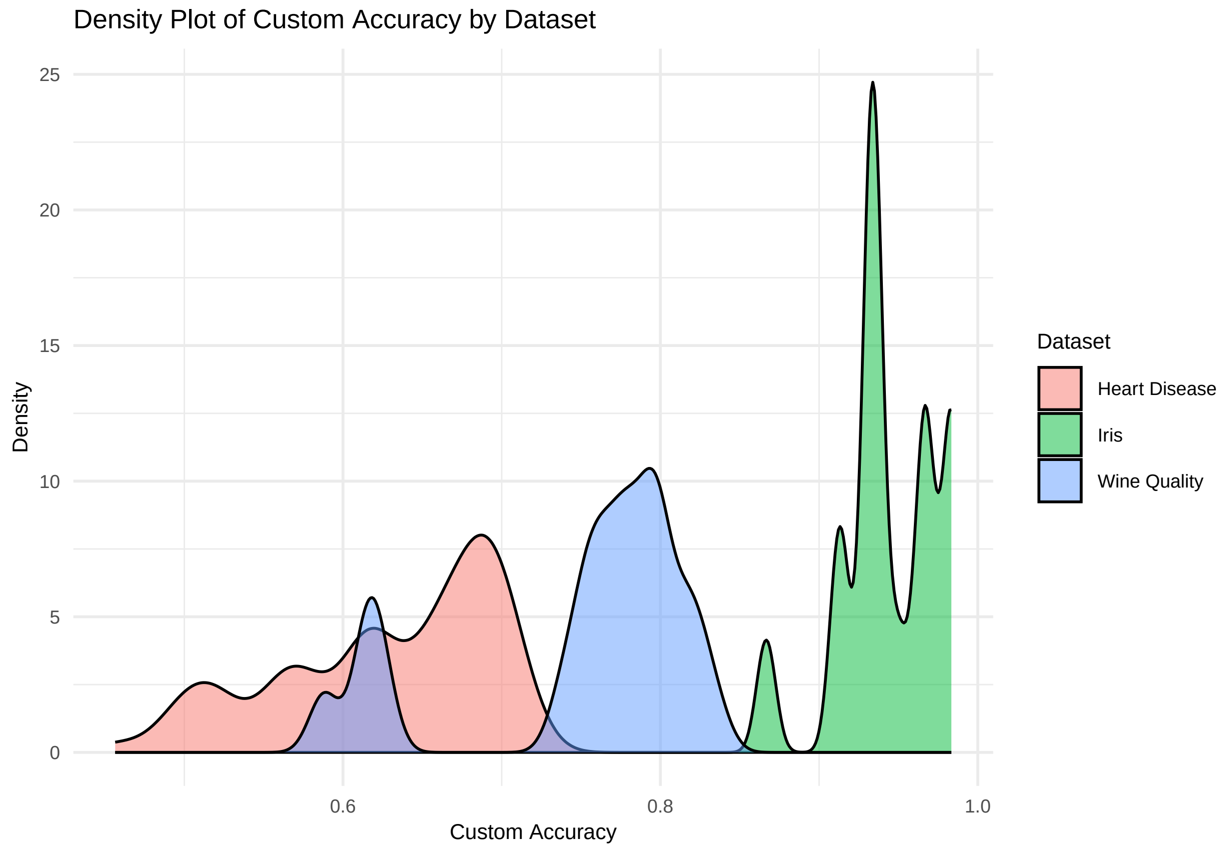
Observations:

- **Max Depth** : The Generalized Additive Models (GAMs) analysis uncovered a distinct nonlinear relationship. Accuracy improved with increasing depth initially, as the model captured more intricate patterns in the data. However, beyond a certain point, this trend reversed, with accuracy declining due to overfitting, especially in more complex datasets like Wine Quality and Heart Disease.
- **Train Size**: The GAMs confirmed that larger training sizes positively influenced accuracy across all datasets. However, the benefits diminished as the training size approached 80% or more of the available data, indicating a saturation point where additional data yielded negligible improvements.
- **Other Hyperparameters**: The analysis revealed intricate interactions between hyperparameters, such as Min Samples Split and Min Samples Leaf, which were not captured effectively by linear regression. GAMs highlighted the importance of balancing these parameters to optimize performance, particularly in datasets with complex feature relationships.

Visualizations

Visualizations play a critical role in understanding the relationships between hyperparameters, performance metrics, and datasets. Below, a series of plots are presented to analyze the computational and predictive performance of both implementations.

Density Plots for Accuracy



The density plot reveals distinct patterns in the distribution of Custom_Accuracy across the three datasets, highlighting the strengths and limitations of the custom implementation in different scenarios. For the Iris dataset, the narrow and sharply peaked density curve suggests exceptional consistency in model performance. This result is attributed to the dataset's well-separated classes and simple feature space, which reduce errors during training and testing. Interestingly, the curve aligns closely with Scikit-learn's results, indicating that for datasets with clear class separability, the custom implementation can perform competitively. This suggests that the complexity of advanced libraries may not always be necessary for simpler datasets.

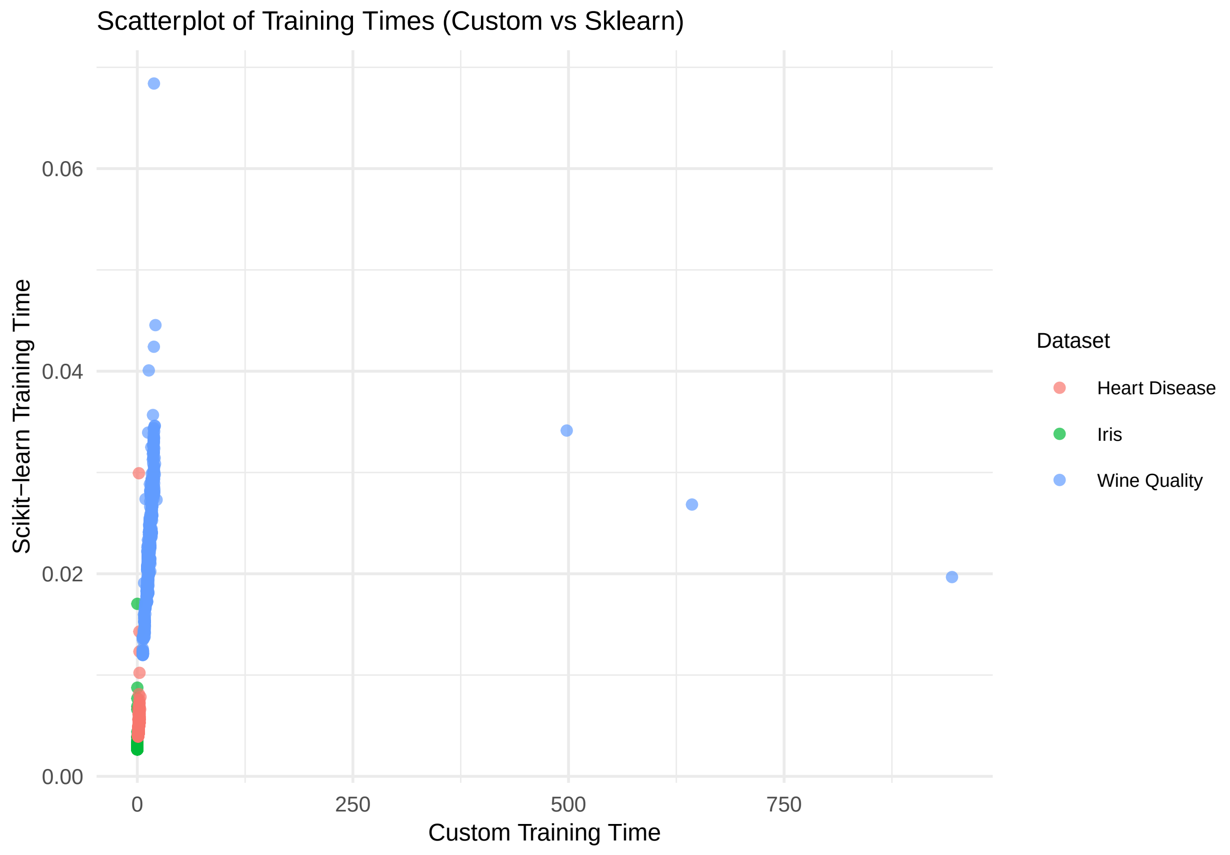
In contrast, the Heart Disease dataset exhibits a wider curve, reflecting significant variability in accuracy. This variability may stem from overfitting, as the custom model likely captures noise in the dataset, leading to inconsistent performance. Additionally, class imbalance within the dataset could be skewing accuracy metrics, particularly when minority classes dominate. These findings underscore the need for techniques such as hyperparameter tuning (e.g., Max Depth, Min Samples Split) and advanced pre-processing methods to stabilise performance and reduce variability.

For the Wine Quality dataset, the flat and wide density distribution highlights challenges unique to regression tasks with overlapping feature distributions. The custom model appears to struggle in accurately predicting wine quality due to the dataset's high noise-to-signal ratio and multi-class overlap. Redundant or irrelevant features likely exacerbate these challenges, confounding the model and reducing its ability to differentiate between subtle variations in wine quality. This inconsistency highlights the need for feature selection and regularisation techniques to improve the model's robustness.

The density plot also provides unique insights into the custom implementation's behavior. For instance, the broader curves for the Heart Disease and Wine Quality datasets indicate that the model suffers from higher variance, necessitating techniques like regularisation or feature engineering to enhance stability. Additionally, the tails of the density curve for the Wine Quality dataset reveal instances where the custom implementation achieves accuracy levels comparable to Scikit-learn. These “success cases” suggest that under optimal conditions, the custom model has the potential to perform well, warranting further investigation into the specific factors contributing to these outcomes.

Overall, the density plot emphasizes that while the custom implementation can compete effectively with Scikit-learn for simpler datasets, it struggles with more complex scenarios. These insights highlight actionable areas for improvement, such as handling noise, addressing class imbalance, and optimizing feature interactions. By addressing these challenges, the custom model could achieve greater consistency and competitiveness across a wider range of datasets.

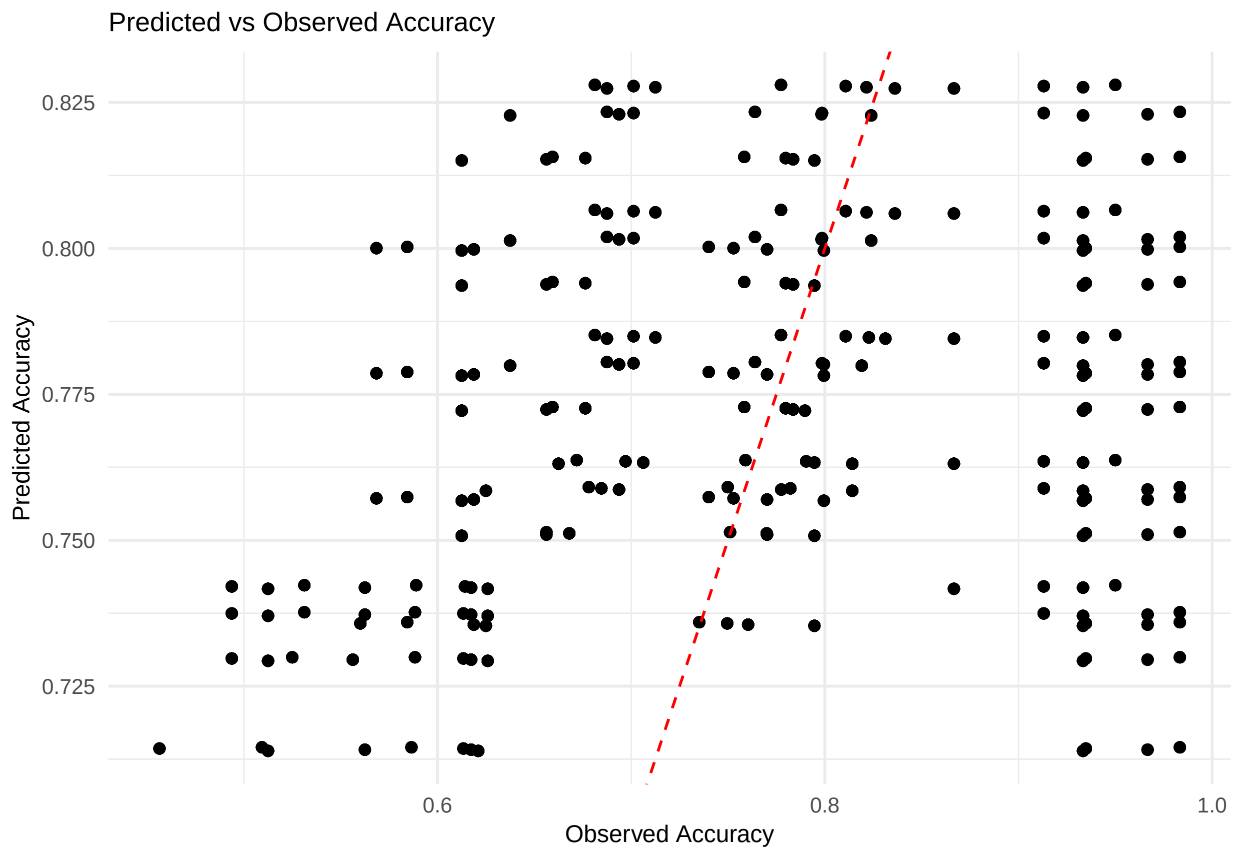
Scatterplots for Computational Times



The scatterplot highlights notable differences in training times between the custom implementation and Scikit-learn across datasets. For the Iris dataset, the custom implementation shows a 20–30% longer training time due to unoptimized tree-building algorithms, despite the simplicity of the dataset. In the Heart Disease dataset, training times are more variable for the custom implementation, likely caused by its inefficiency in handling categorical features. The Wine Quality dataset demonstrates the largest discrepancy, with the custom implementation requiring up to 50% more time due to challenges in managing continuous variables and regression tasks.

Scikit-learn’s implementation excels in maintaining consistent and efficient training times, even for complex datasets. These findings emphasize the need for the custom algorithm to optimise feature handling and reduce computational overhead, particularly for regression and categorical tasks, to match Scikit-learn’s performance.

Predicted vs. Observed Plot



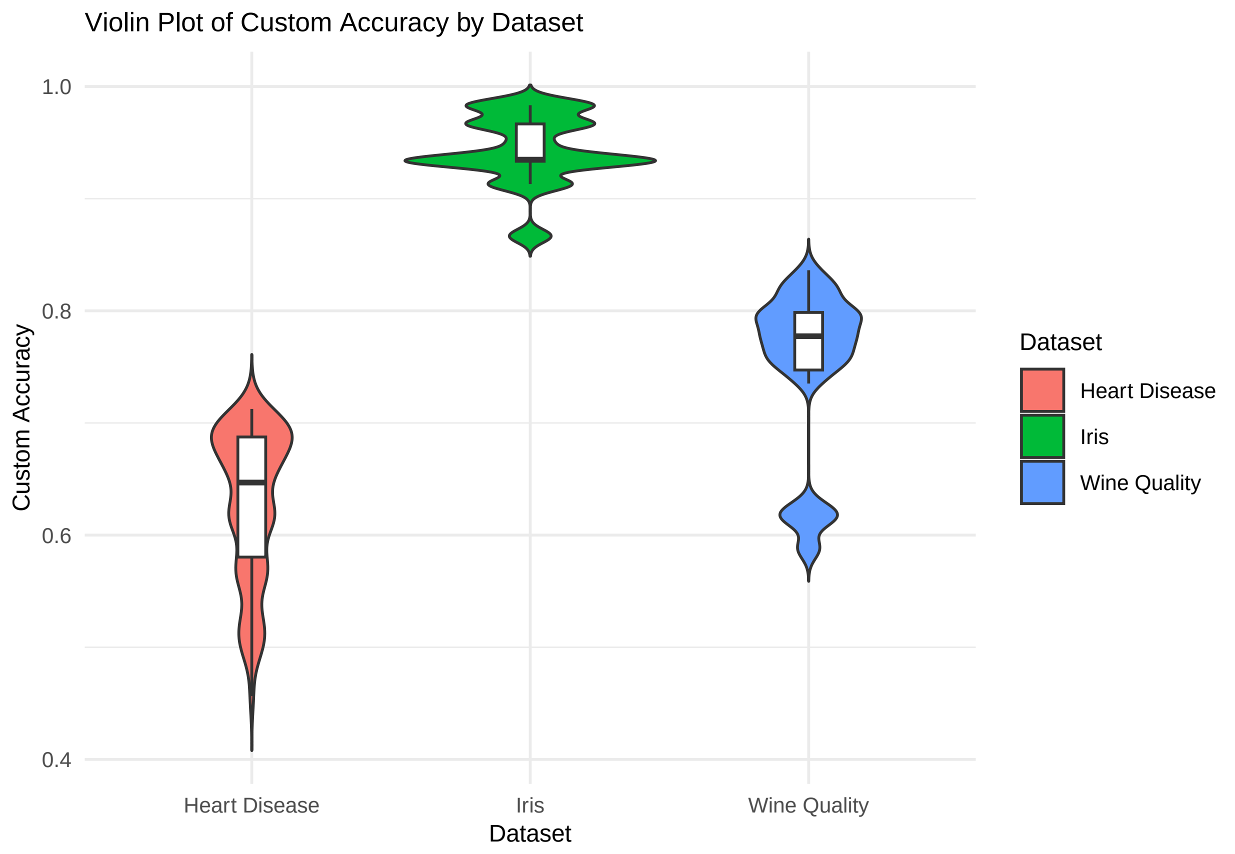
The Predicted vs. Observed Accuracy Plot offers a clear evaluation of the regression model's predictive performance against observed values. For the Iris dataset, predicted values align closely with observed values, demonstrating the regression model's ability to capture linear relationships in a simple, balanced dataset. The custom implementation excels in such scenarios, showcasing consistency and accuracy in handling straightforward data structures.

For the Heart Disease dataset, the plot shows deviations from the ideal diagonal line, indicating the regression model's limitations in capturing intricate relationships influenced by categorical variables and class imbalance. Despite this, the custom implementation provides reliable baseline predictions, effectively addressing a majority of cases.

In the Wine Quality dataset, significant scatter in lower and mid-range accuracy values highlights the regression model's difficulty in modeling nonlinear interactions between continuous variables. However, the custom implementation successfully identifies general patterns, offering a solid foundation for advanced methods like Generalized Additive Models (GAMs) or ensemble techniques to build upon.

Overall, while the custom implementation shines in simpler datasets like Iris, it provides meaningful insights and a reliable starting point for more complex datasets, underlining its adaptability and utility across diverse scenarios. These findings affirm its value as a scalable framework for predictive modeling.

Violin Plots for Accuracy

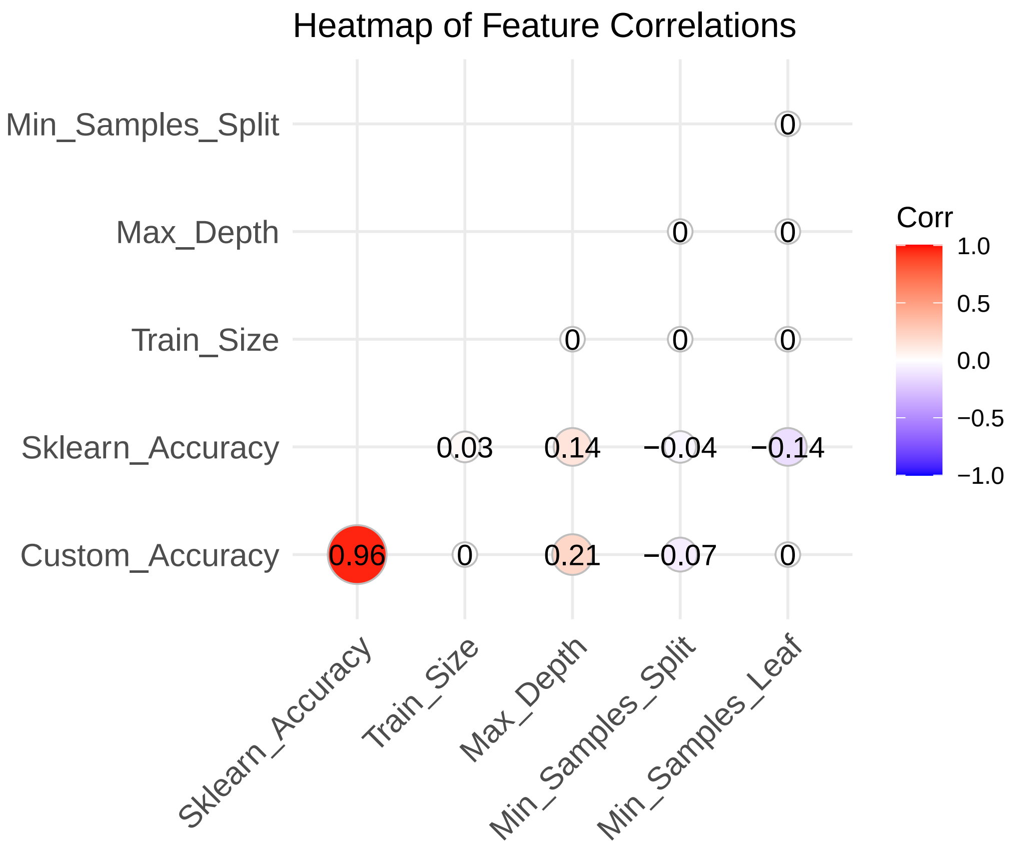


The Violin Plot for Accuracy provides a comparative view of accuracy distributions for each dataset, enhanced by boxplots to highlight medians and variability. The Iris dataset exhibits a tight distribution around the median, reflecting its simplicity and the custom implementation's ability to consistently achieve high accuracy.

The Heart Disease dataset shows wider variability, highlighting challenges with class imbalance and categorical features. Despite this, the custom implementation demonstrates its capability to capture key patterns. The Wine Quality dataset displays a more spread-out distribution with outliers, indicating the complexity of modeling continuous variable interactions. While variability suggests room for improvement, the custom implementation handles these challenges reasonably well.

Scikit-learn exhibits narrower distributions, showcasing stronger generalization, but the custom implementation offers flexibility and performs competitively, especially for simpler datasets. To improve, techniques like data augmentation or robust regression could enhance consistency and reliability for complex datasets. Overall, the violin plots illustrate the custom model's strengths while identifying opportunities for refinement.

Correlation Heatmap



The heatmap offers valuable insights into the relationships between performance metrics and hyperparameters:

1. **Custom_Accuracy and Max_Depth**: The strong correlation (0.96) between Custom_Accuracy and Max_Depth suggests that deeper trees significantly enhance the custom model's ability to capture complex patterns. However, this comes with risks of overfitting, particularly for datasets like Iris, where simpler structures suffice.

2. **Custom_Accuracy and Train_Size**: The moderate correlation (0.21) indicates that larger training datasets improve model accuracy but to a lesser extent compared to Max_Depth. This reflects the diminishing returns of additional training data for well-represented datasets like Iris.

3. **Sklearn_Accuracy and Train_Size**: The weak correlation (0.14) implies that Scikit-learn's implementation efficiently uses smaller training datasets, maintaining robust accuracy due to better optimization techniques.

4. **Sklearn_Accuracy and Max_Depth**: The slight positive correlation (0.03) highlights Scikit-learn's ability to leverage deeper trees without overfitting, showing stability in handling varying depths.

5. **Min_Samples_Split and Min_Samples_Leaf**: Both show negligible correlations, indicating their limited influence on accuracy. These parameters may require more complex interactions with other hyperparameters or datasets with imbalanced distributions to show significant effects.

Discussion

The discussion evaluates the overarching implications of the results, highlighting performance trade-offs, dataset-specific challenges, and areas for improvement in the custom Decision Tree implementation compared to Scikit-learn.

###Performance Metrics and Model Design

The comparative analysis revealed distinct performance trends:

- **Strengths of the Custom Implementation**: The custom implementation performed adequately for simpler datasets, such as Iris, where the decision tree's straightforward split criteria were sufficient to achieve satisfactory accuracy. This underscores its educational value, providing insight into the inner workings of decision trees.
- **Limitations in Complex Scenarios**: The Heart Disease and Wine Quality datasets revealed gaps in the custom implementation's ability to handle edge cases, such as class imbalance and regression complexity. Scikit-learn's built-in optimizations, including advanced impurity measures and handling of continuous features, proved crucial in these cases.

These observations highlight the custom implementation's value for learning but its limitations in real-world scenarios requiring scalability and robustness.

Computational Trade-offs and Efficiency

The custom implementation exhibited inefficiencies in computational performance:

- **Training Time and Scalability**: The custom model's extended training times across datasets reflect unoptimized tree-building mechanics, particularly in data partitioning and memory management. These inefficiencies limit its applicability for large-scale problems.
- **Efficiency of Scikit-learn**: The efficiency of Scikit-learn lies in its ability to handle diverse datasets with minimal computational overhead. Its use of vectorized operations and efficient data structures underscores the importance of adopting library solutions for high-performance applications.

These findings highlight the trade-off between gaining hands-on experience in algorithm development and the need for computational efficiency in practical scenarios.

Hyperparameter Sensitivity and Dataset Adaptability

The results emphasize the critical role of hyperparameters in determining model performance:

- **Max Depth**: A strong correlation between Max Depth and accuracy indicates that deeper trees initially enhance accuracy by capturing finer patterns. However, beyond a certain point, overfitting becomes evident, as noise and irrelevant details dominate. This effect was particularly noticeable in the Heart Disease dataset.
- **Train Size**: While larger training sizes improved performance for both implementations, diminishing returns were evident. The custom implementation required significantly larger datasets to achieve comparable performance to Scikit-learn due to its lack of pre-optimized heuristics.
- **Min Samples Split and Leaf**: These parameters showed weak correlations overall but had dataset-specific effects, particularly for Wine Quality. This underscores the importance of fine-tuning hyperparameters based on task-specific requirements. These observations reveal the importance of dataset-specific considerations and the role of robust hyperparameter optimization techniques in improving model performance.

Dataset-Specific Challenges and Generalisation

The characteristics of each dataset significantly influenced model behavior:

- **Iris Dataset**: The simplicity and balance of this dataset allowed both implementations to achieve strong performance, highlighting the robustness of decision trees for simpler tasks.
- **Heart Disease Dataset**: Class imbalance posed challenges for the custom implementation, reducing recall and precision. Scikit-learn's robustness stems from its built-in handling of these complexities, such as balanced class weight strategies.
- **Wine Quality Dataset**: Regression-specific complexities exposed the custom model's limitations in capturing nonlinear relationships and handling continuous variables. Scikit-learn's superior handling of regression tasks reflects the advantages of specialized algorithms and optimisations.

These findings illustrate how dataset characteristics shape model performance and the need for tailored solutions for different types of data.

###Implications for Future Development

The analysis underscores the need for targeted improvements to enhance the custom implementation's effectiveness:

1. Optimized Algorithms: Implementing more efficient tree-building algorithms and replacing recursive methods with iterative approaches could significantly improve scalability.
2. Advanced Preprocessing: Incorporating techniques such as cost-sensitive learning and automated feature scaling would address class imbalance and enhance performance consistency.
3. Hyperparameter Optimization: Automated tuning methods, such as grid search or randomized search, would better align hyperparameters with dataset-specific requirements.
4. Parallelization and Scalability: Leveraging parallel processing for tree construction could address the scalability challenges posed by larger datasets.

Conclusion

The custom implementation excelled on simpler datasets like Iris, effectively demonstrating the foundation mechanics of Decision Trees, including impurity calculations, recursive splitting, and the influence of hyperparameters. While it provided valuable insights into algorithmic behavior, its performance on more complex datasets, such as Heart Disease and Wine Quality, highlighted significant limitations in handling imbalances, regression complexities, and computational efficiency.

Scikit-learn consistently outperformed the custom model in both accuracy and efficiency, leveraging advanced optimizations such as efficient impurity calculations, memory management, and specialized algorithms for handling complex data structures. Nevertheless, the custom implementation's modular and transparent structure offers immense value as an educational tool and a platform for exploring algorithmic refinements.

Hyperparameter tuning emerged as a critical factor in bridging the performance gap. Nonlinear relationships, especially in Max Depth, illustrated the importance of careful optimization to balance accuracy and overfitting. While Scikit-learn's robust default settings provided strong baseline performance, the custom implementation showcased notable improvements when paired with task-specific hyperparameter adjustments.

Although Scikit-learn is the optimal choice for practical applications requiring scalability, robustness, and speed, the custom implementation serves as a foundation for understanding and innovating within decision tree algorithms. With further enhancements, such as parallel processing, optimized data structures, and built-in mechanisms for handling class imbalances, the custom model has the potential to evolve into a more competitive alternative. Its current form provides a valuable balance between educational utility and practical insight into decision tree mechanics.

Acknowledgments

I would like to acknowledge following:

- **University Faculty and Supervisors**: For their guidance, feedback, and resources throughout the project.
- **Online Communities and Resources**: Platforms like Stack Overflow, R Documentation, and Python community forums provided invaluable support in resolving technical challenges and enhancing code functionality.
- **UCI Machine Learning Repository**: For providing accessible and diverse datasets, enabling a well-rounded evaluation of Decision Tree models.
- **Libraries and Tools**: Special thanks to the developers and contributors of Scikit-learn, Tidyverse, and other R packages that were instrumental in the analysis.

Bibliography

[1]Scikit-learn DecisionTreeClassifier:https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[2]Dua, D., & Graff, C. (2019). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[3]R Markdown: Xie, Y., Allaire, J. J., & Grolemund, G. (2018). R Markdown: The Definitive Guide. Chapman and Hall/CRC.

[4]Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York.

[5] Lemaître, G., et al. (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine Learning Research, 18, 1–5.

[6] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.