A Project Report

On

"Evaluating Efficiency, Accuracy, and Parameter Sensitivity Across UCI Datasets".

Comparative Analysis of Custom Decision Tree Implementation and Scikit-learn: Computational and Machine Learning Perspectives on Breast Cancer

Submitted to

Lancaster University

In Partial Fulfillment of the Requirement for the Award of

MASTER'S DEGREE IN

<Data Science>

36851079

Under the Guidance Of:
Prof. Jun Liu.

Prof. Christopher Jewell.

Section I: ABSTRACT

The report explains how the decision tree works and the concept of Gini Index as well. This report explains the implementation and evaluation of a Decision Tree classifier in Python and R, and its comparison with the Scikit-learn library's implementation. Datasets from the UCI Machine Learning Repository were used to evaluate computational and machine learning performance metrics. Parameters such as tree depth and training set size were varied to analyze their impact on accuracy, precision, recall, F1 score, and training time. The analysis of results was performed using R for statistical evaluation and visualization. The datasets focused on are Iris Dataset, Wine Dataset and Breast Cancer Dataset. Using of three different datasets helped with more in-depth analysis.

Section II: INTRODUCTION.

The main objective of this coursework is to build a custom decision tree classifier and compare it with Scikit's Decision Tree classifier. Three different datasets, are used to fit to the custom made decision tree classifier built from scratch and compare their performance with the library implementation. This part of the report explains the concepts in detail. Followed by the objectives and methodologies used for the completion of this coursework.

A) Decision Trees: The Decision Tree is a widely used supervised learning algorithm due to its interpretability and efficiency. The name itselt suggests that it is used for making decisions from the given dataset. A decision tree splits the dataset into subsets based on feature values. The splitting continues until a stopping criterion is met, the leaf nodes are "pure". The structure of a decision tree usually includes: a) Root Node – which represents the entire dataset and the first decision, b) Internal Nodes: Which represent the outcome of a decision, c) Edges: which represent the outcome of a decision and d) Leaf Nodes: which represent the final classification.

B) Splitting Criteria: At each step, the Decision Tree algorithm selects a feature to split the data. The selection is based on measures of impurity or how well a split separates the classes. Some common splitting criteria are: Entropy: Also known as Information Gain. In the context of decision trees, entropy can be – how much variance the data has. It is used in ID3 and c4.5 algorithms. It measures the reduction in uncertainty after a split. Gini Index: The Gini index also known as Impurity, calculates the likelihood that somehow a randomly picked instance would be erroneously catalogued. It is a measure of how impure or mixed a dataset is. It ranges between 0 and 1, where 0 is a pure dataset and 1 is a completely impure dataset. This indicates that an attribute with a lower Gini index should be preferred.

C) Datasets: The following three datasets from UCI Repository were used to complete the coursework:

• Iris Dataset : A small classic dataset from Fisher, 1936. One of the earliest known datasets used for evaluating classification methods.

• Breast Cancer Dataset (Breast Cancer Wisconsin (Diagnostic)): Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

• Wine Quality Dataset: Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests.

D) Objectives • Implement a Decision Tree classifier using the Gini Index. • Compare its performance with Scikit-learn's implementation. • Analyze the impact of varying hyperparameters and dataset sizes. • Perform statistical analysis using R.

Section III: METHODOLOGY

3.1 Loading the Data:

The first step was to load the datasets into the python notebook. The datasets were acquired from UCI Machine Learning Repository. Used the print function to print the feature names of the dataset.

3.2 Decision Tree Algorithm:

The custom Decision Tree was implemented using the Gini Index as the splitting criterion. The algorithm includes: • Recursive partitioning of data based on feature thresholds. • Pruning mechanisms to control overfitting. • Hyperparameter options such as maximum tree depth.

The Scikit-learn implementation ("DecisionTreeClassifier") was used as a benchmark. I. The implementation of the decision tree adopts a systematic method to recursively divide the dataset, reducing impurity at every stage using the Gini Index as the criterion for splitting. Important aspects of this implementation include:

1. Gini Index Calculation: The `calculate_gini` function evaluates the impurity of potential splits by aggregating the proportions of each class present in the left and right child nodes, thus assessing the separation of classes. A lower Gini value signifies a more effective split.
2. Data Splitting: The `split_data` function splits the dataset into left and right subsets based on specified feature index and threshold value. This step serves as the foundation for assessing potential splits.
3. Finding the Best Split: The `get_best_split` function reviews all features and unique values within the dataset to determine the split that results in the lowest Gini Index. This function returns the optimal feature index, split value, and the associated groups.
4. Tree Construction: The `build_tree` function initiates the recursive process of tree construction. It sets up the root node using the best split and calls the `split_node` function to continue dividing the dataset until a stopping criterion (such as maximum depth or minimum node size) is reached.
5. Terminal Node Creation: The `to_terminal` function identifies the predicted class for a terminal node by selecting the most common class in the subset of the dataset associated with that node.
6. Recursive Splitting: The `split_node` function manages the recursive splitting process. If a node satisfies the stopping conditions (like depth or size), it is classified as a terminal node. Otherwise, the splitting process persists with additional divisions.
7. Custom Decision Tree Class: The `DecisionTreeGini` class encompasses the logic for building the tree and making predictions: The `fit` method constructs the tree from the training data and the `predict` method navigates through the tree to produce predictions for the test data.

This implementation creates a binary decision tree with user-specified limits on maximum depth and minimum node size. It replicates the functions of conventional decision tree classifiers, offering a basis for comprehending the workings of tree-based models.

\end{enumerate}

3.3 Datasets

The following datasets from the UCI repository were used:

- Iris: Classification of flower species (150 instances).
- Breast Cancer: Binary classification of cancer diagnosis (569 instances).
- Wine: Classification of wine quality (178 instances).

3.4 Experimental Design

This part of the coursework involves using different evaluation metrics to evaluate the decision tree classifier.

- **Metrics Used:**

The different metrics used are:

o **Accuracy** – Accuracy gives us the number of correctly classified data instances over the total number of data instances.

o **Precision** – Precision should ideally be 1 for a good classifier. It tell us out of all the positive predictions we made how many were true.

o **Recall** – It focuses on how good the model is at finding all the positives.

o **F1 Score** – It is a measure that combines recall and precision.

o **Training Time** – It is the time taken by a model to train on a dataset.

• **Hyperparameter Variations:** Hyperparameters directly control model structure, function and performance. Tuning them allows us to tweak model performance for optimal results.

o Maximum Tree Depth: 5, 10, 15.

• **Dataset Splits:** It is a process that involves randomly dividing a dataset into two subsets. The training set is used to train a model and test set is used to evaluate the model's performance. In the implementation we have split the dataset in the following ratio:

o Training: 80%

o Testing: 20%

3.5 Python Implementation

The custom Decision Tree was implemented in Python and compared to Scikit-learn. The decision tree implementation was used on all three datasets. And their performance was compared with the Scikit-learn library implementation. Intermediate results were saved as CSV files for further analysis in R.

3.6 Visualization

- **Accuracy vs. Tree Depth**: Line plots demonstrated accuracy trends.
- **Training Time Comparison**: Bar charts highlighted computational differences.

Section IV: RESULTS

4.1 PYTHON Results

Detailed Analysis of Results

In this part I present the results obtained in this coursework. The performance of the custom decision tree model and the sklearn decision tree model was assessed on three datasets: Iris, Breast Cancer, and Wine. The primary performance metrics—accuracy, precision, recall, F1 score, and training time—were computed for both models over these datasets.

Iris Dataset
Custom Decision Tree: The custom decision tree model achieved outstanding results on the Iris dataset, obtaining accuracy, precision, recall, and F1 score of 1.0. The training duration was 0.11 seconds.
Sklearn Decision Tree: The sklearn decision tree model also delivered perfect results, scoring 1.0 across all metrics. The training time was notably quicker, lasting just 0.001 seconds.

Breast Cancer Dataset
Custom Decision Tree: The custom decision tree model reached an accuracy of 93.86%, with precision and recall both at 93.90% and 93.86%, respectively. The F1 score was recorded at 93.87%. The model required 10.95 seconds to train.
Sklearn Decision Tree: The sklearn model exhibited slightly superior performance, achieving an accuracy of 94.74%, alongside precision and recall rates of 94.74%, and an F1 score of 94.74%. The training time was 1.59 seconds.

Wine Dataset
Custom Decision Tree: The custom decision tree model recorded a lower performance on the Wine dataset, registering an accuracy of 55.63%, precision of 52.28%, recall of 55.63%, and an F1 score of 53.36%. The training time was 55.73 seconds.
Sklearn Decision Tree: The sklearn model showed marginally better performance with an accuracy of 55.94%, precision of 52.58%, recall of 55.94%, and an F1 score of 53.70%. The training duration was significantly shorter at just 0.005 seconds.

4.1.1 Computational Evaluations

Training Time:

o The Scikit-learn implementation was consistently faster. The custom implementation exhibited exponential increases in training time with tree depth.

o The implementation in Scikit-learn was notably quicker than the custom version. Specifically, the custom implementation required a considerable amount of time to train, particularly on the Wine dataset, which took 55.73 seconds in contrast to just 0.005 seconds for Scikit-learn. This observation corresponds with the finding that the custom implementation showed exponential growth in training time as tree depth increased, particularly with larger datasets like Wine.

4.1.2 Machine Learining Evaluations

Accuracy and Precision:

o Both implementations achieved comparable accuracy, with Scikit-learn slightly outperforming in larger datasets. Scikit-learn slightly surpassed the custom model on the Breast Cancer dataset (94.74% vs. 93.86% accuracy), though the difference was marginal. In the Iris dataset, both models achieved flawless accuracy. Additionally, the variations in precision and recall were minimal across the datasets, indicating that both models were largely equivalent in these metrics, with Scikit-learn showing a slight advantage on larger datasets like Breast Cancer. This supports the initial assertion that accuracy levels are comparable, with a minor lead for Scikit-learn in larger datasets.

o Precision and recall showed minimal differences.

4.1.3 Hyperparameter Tuning Analysis

Enhancing maximum depth: According to the regression analysis, increasing the maximum depth did not meaningfully influence accuracy (p-value = 0.3504), implying that accuracy remained relatively constant across varying depths, contrary to the anticipated overfitting trend. However, training time rose significantly with tree depth, particularly for the custom implementation, aligning with the hypothesis that training time escalates with tree depth, especially in the custom model. The custom model required 55.73 seconds to train on the Wine dataset, while the Scikit-learn model only took 0.005 seconds, highlighting that the custom implementation has a much steeper increase in training time as depth grows, consistent with the idea of exponential growth in training duration.

4.2 R Results

Statistical Analysis Using R

I also analyzed the results from the R environment, where I explored the model's performance. Below is a detailed breakdown of the key findings:

4.2.1 Linear Regression:

A linear regression analysis was performed in R to understand the relationship between maximum tree depth and accuracy. The regression findings revealed that the maximum depth variable did not significantly impact accuracy (p-value = 0.3504), as shown by the non-significant coefficient

for max_depth. The R-squared value was 0.2181, indicating that the model's accuracy could be influenced by other factors not considered in the analysis.

4.2.2 T - Tests

A Welch Two Sample t-test was executed to examine the training durations of the custom and sklearn models. The test indicated a t-value of 1.276, with a p-value of 0.3299. Given that the p-value exceeds the significance threshold of 0.05, we do not reject the null hypothesis, suggesting there is no meaningful difference in the training durations of the two models.

4.2.3 ANOVA Analysis

An Analysis of Variance (ANOVA) was conducted to assess the impact of the dataset and model on the performance metrics. The results showed that the model and dataset factors did not significantly influence the accuracy of the decision trees (p-values of 0.988 and 0.742, respectively). This indicates that the performance of the models was not substantially different across datasets.

Section V DISCUSSION and Conclusion

In this part, I compared the performance of a custom Decision Tree implementation (developed manually in Python) with Scikit-learn's library implementation of Decision Tree classifier, along with a similar analysis performed in R. The outcomes from both programming environments provide important information about the precision, effectiveness, and real-world uses of these models.

5.1 Unexpected Results:

The findings from the Wine Quality dataset were somewhat surprising, as the custom decision tree model managed to achieve a relatively modest accuracy of 55.6%, while the Scikit-learn model reached an accuracy of 55.9%. This slight discrepancy in performance may indicate that the custom implementation faces difficulties in identifying the underlying patterns within the data, especially considering the dataset's complex characteristics, which incorporate both numerical and categorical elements. Additionally, the custom model showed a significant increase in training duration (over 55 seconds), revealing possible inefficiencies in its algorithm in contrast to the streamlined Scikit-learn model, which was able to complete training in just a few milliseconds. These results emphasize the difficulties of constructing decision trees from the ground up, particularly when working with real-world, multi-dimensional datasets like Wine Quality.

5.2 Limitations

There are several limitations to this study that should be considered. First, the custom decision tree implementation is relatively basic and lacks the optimizations found in more sophisticated libraries like Scikit-learn. This led to slower training times and potentially less efficient handling of large datasets, as seen in the Wine Quality dataset. Additionally, the evaluation metrics were limited to accuracy, precision, recall, and F1 score, which may not fully capture the performance nuances of the models, especially in imbalanced datasets.

5.3 Discussion

The comparison revealed that while the custom implementation is a valuable learning exercise, Scikit-learn's implementation is more practical for real-world applications due to its efficiency and reliability. The analysis underscored the importance of balancing model complexity and computational constraints. The evaluation of the custom and Scikit-learn Decision Tree models reveals several key findings regarding their performance and computational efficiency. Both models attained high accuracy across the datasets, with the Scikit-learn model marginally surpassing the custom model on larger datasets, such as Breast Cancer. Nevertheless, the discrepancies in accuracy, precision, and recall were minimal, suggesting that both models are largely similar in their effectiveness at classifying data.

Regarding training time, Scikit-learn exhibited a significant advantage. The custom model experienced a notable increase in training time, especially with larger datasets like Wine. This observation aligns with the assumption that the custom implementation is less optimized, resulting in slower performance

as tree depth increases. Training time for the custom model also grew exponentially, as illustrated by the Wine dataset (55.73 seconds), compared to just 0.005 seconds for Scikit-learn.

Overall, these results emphasize the efficiency of Scikit-learn in terms of both performance and computational resources, especially when managing larger datasets. Although the custom implementation offers comparable accuracy and precision, it is hindered by longer training times, which may restrict its scalability for larger and more intricate datasets. Future research might aim to enhance the custom implementation to decrease training times and boost scalability, perhaps by integrating more efficient algorithms for tree construction and pruning.

Section VI CONCLUSION

This project demonstrated the implementation and evaluation of Decision Tree classifiers. The custom implementation provided insights into the algorithm's mechanics, while Scikit-learn offered a performance benchmark. Statistical analysis in R reinforced the findings and provided actionable insights for model optimization. Both custom and sklearn decision tree models performed well on the Iris and Breast Cancer datasets, achieving high accuracy and other performance metrics. The Wine dataset, however, presented a more challenging problem, with lower scores for both models. Statistical analyses confirmed that there were no significant differences in training times between the custom and sklearn models, and the maximum depth did not have a substantial effect on model performance. Overall, both models demonstrated similar behavior across datasets, with training time being the main differentiating factor.

Bibliography

Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research.

UCI Machine Learning Repository. Available at: http://archive.ics.uci.edu/ml/index.php

Wickham, H. (2016). ggplot2: Elegant Graphics for Data Analysis. Springer.

Scikit-learn Decision Tree Documentation

Tutorial: How to implement the Decision Tree From Scratch in Python by Machine Learning Mastery. https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

Geeks For Geeks - https://www.geeksforgeeks.org/gini-impurity-and-entropy-in-decision-tree-ml/

B. T. Jijo and A. M. Abdulazeez, "Classification based on decision tree algorithm for machine learning," Journal of Applied Science and Technology Trends

Rho, D., & Lee, J. (2023). Superfast Selection for Decision Tree Algorithms. arXiv. https://arxiv.org/abs/2405.20622.

Hoxhaj, D. (2020). Implementing a Custom Decision Tree Classifier from Scratch. Medium. Implementing a Custom Decision Tree Classifier from Scratch | by Diellorhoxhaj | Jan, 2025 | Medium

Campos, M. (2020). Decision Tree Implementation from Scratch + Visualization. Medium. https://medium.com/%40omidsaghatchian/decision-tree-implementation-from-scratch-visualization-5eb0bbf427c2