Adaptive Decision Trees with Dynamic Depth Pruning for Uneven Data Distributions

Aarav Gupta, Rohith Yelisetty

January 28, 2025

Abstract

Many machine learning datasets exhibit imbalanced distributions, with dense and sparse regions that challenge traditional decision tree models. Conventional decision trees tend to overfit in dense areas while underfitting in sparse regions, limiting their generalizability. This paper introduces an adaptive decision tree algorithm that dynamically adjusts its depth based on local data density. By incorporating a density-based pruning mechanism inspired by K-Nearest Neighbors (KNN), the proposed method prevents excessive branching in sparse regions while allowing deeper splits in dense areas, striking a balance between interpretability and adaptability. Experiments on a traffic accident prediction dataset demonstrate that our model achieves higher generalization performance compared to standard decision trees and pruned variants, with a 20% improvement in test accuracy and a 0.35 improvement in macro-averaged precision, recall, and F1-Score. These findings suggest that density-aware decision trees offer a promising approach for handling uneven data distributions in domains such as healthcare, finance, and autonomous systems.

1 Introduction

Many datasets in machine learning are imbalanced. Parts of a dataset may contain a lot of closely packed data points, known as dense regions, while other parts may contain very few scattered data points, known as sparse regions. Usually, decision trees are simple and interpretable but they don't handle these kinds of datasets very well. In the presence of dense regions, they tend to grow too deep and overfit to small patterns that do not generalize well. In sparse regions, they might turn out too shallow and miss the meaningful relationship.

The project tries to solve these problems by developing an adaptive decision tree that may change its depth dynamically with regard to the density of data at each node. If the data in a region is dense, the tree will allow deeper splits to capture finer patterns. In the case that the data is sparse, it will self-prune early in order to avoid overfitting to noise. This method brings together the global structure from decision trees with the local adaptability of KNN. The result is a way more intelligent tree which can handle datasets that are unevenly distributed. It's very useful for fields such as healthcare, finance, and real estate since data distributions are not uniform most of the time.

The input to our algorithm is a mix of quantitative and qualitative data, containing various factors affecting road conditions, driver behavior, and traffic situations. We then use a decision tree integrated with Nearest Neighbors to output the predicted severity of traffic accidents.

2 Related Work

Research on imbalanced datasets in decision trees has yielded a number of promising approaches, each with its strengths and weaknesses. One of the most interesting strategies is that of the dynamic adjustment of tree depth with respect to data density, as in [1]. Their method effectively presents overfitting in dense regions by limiting unnecessary splits while reducing underfitting in sparse regions by maintaining computational efficiency. However, one major limitation lies in the loss of local adaptability since their method considers only global adjustments. In contrast, our algorithm integrates density-based depth control with localized adaptability to capture finer patterns and is thus more flexible and generalizable across a wide range of datasets.

The other related line of research relies on local adaptability via KNN-inspired techniques, as in [2]. These methods combine decision tree frameworks with neighborhood-based adjustments and are good

at picking up local patterns. On the other hand, they tend to be computationally expensive, in particular for large data sets. While the introduction of localized adjustments was a very clever and successful idea, these methods often sacrifice global interpretability in favor of accuracy. Our work addresses this trade-off by finding the right balance between local adaptability and global interpretability with a solution which is scalable and comprehensible at the same time.

Hybrid pruning techniques for decision trees, some examples of which may be seen in [3], proposed optimizations in their structures by removal of branches using various threshold mechanisms. While such methods certainly achieve better computational efficiency with reduced overfitting, still, there remains substantial need for tedious parameters tuning, and this is notoriously subject to human fallacies. As stated earlier, automatic pruning methods involve data density measures themselves, dispensing with human effort while making it rather practical in wide applications.

The application of adaptive decision tree models to real-world domains, like healthcare or finance, is another direction where the challenges posed by imbalance datasets have to be urgently addressed. As an example, [4] analyzes the use of adaptive depth control in view of enhancing diagnosis accuracy when medical datasets are sparse. While effective for domain-specific data, these are often not generalizable and usually perform poorly for datasets with high variability. In comparison, our work is designed to be adaptable to various datasets and domains, making it more general while not necessarily sacrificing performance.

Recent advances in decision trees include the state-of-the-art use of deep learning techniques to allow for both scalability as well as robustness [5]. However, most of these hybrid approaches sacrifice the interpretability of the trees. In our opinion, this is a crucial strength of decision trees which shouldn't be easily sacrificed. This work fills the gap of providing a robust yet user-friendly model in this domain by maintaining this feature while adding adaptability.

3 Dataset and Features

Our performance analysis was done based on a dataset taken from Kaggle: *Traffic Accident Prediction Dataset* [6], which focused on predicting traffic accident severity using categories like Low, Moderate, and High. Various features in the dataset include weather condition, road type, time of day, speed limits, the number of vehicles involved, road conditions, vehicle type, driver age, driver experience, and road light conditions. These features were preprocessed to a great extent to prepare the data for analysis.

Weather	**Road Type**	**Time of Day**	**Speed Limit**	**Number of Vehicles**
• Clear → 0 • Rainy → 1 • Foggy → 2	• Highway → 0 • City Road → 1 • Rural Road → 2	• Morning → 0 • Afternoon → 1 • Evening → 2	• (-inf-75.75] → 0 • (75.75-121.5] → 1 • (167.25-inf) → 2	• (-inf-4.25] → 0 • (4.25-7.5] → 1 • (7.5-10.75] → 2

Road Condition	**Vehicle Type**	**Driver Age**	**Driver Experience**	**Road Light Condition**
• Dry → 0 • Under Construction → 1 • Wet → 2	• Bus → 0 • Truck → 1 • Car → 2	• (-inf-30.75] → 0 • (30.75-43.5] → 1 • (43.5-56.25] → 2	• (-inf-24] → 0 • (24-39] → 1 • (39-54] → 2	• Daylight → 0 • Artificial Light → 1 • No Light → 2

Table 1: Feature encoding used in the dataset.

First, the attributes such as speed limit, number of vehicles, driver age, and driver experience were discretized into defined ranges. Then, instances with missing class values were removed, and other missing values were replaced by their respective modes. All the data values were then changed to numeric categories for computation such as Euclidean Distance, in which proximity reflects similarity — clear weather is closer to rainy weather than icy or snowy weather. It follows the conversion done in the numeric encoding, where the clear weather is 0, rainy is 1, foggy is 2, snowy is 3, and stormy is 4. The same consideration has been taken for other attributes like road type, time of day, road condition, and vehicle type.

Our dataset was split into a stratified train-test set with a 70-30 ratio to preserve the class distribution across the subsets. Additionally, all integer values were replaced with strings to ensure

compatibility with the decision tree algorithm. By combining these preprocessing steps, the dataset was optimized for machine learning models while maintaining its integrity and consistency.

4 Methods

Our study is based on the application of a few machine learning algorithms: a regular decision tree, the J48 Pruned Tree from Weka, the Nearest Neighbors algorithm from sci-kit-learn, and a new density-based decision tree we developed for this project. A normal decision tree was used for comparison. It uses feature selection for the best separation of data and splits into partitions, usually done by measures such as InfoGain or Gini impurity. This is a recursive process, continuing to branch until every node represents one class or reaches a previously set stopping condition. Though this is straightforward and works for most datasets, it has the tendency to overfit when there is noise or sparsity in the data. It's memorizing the training data rather than generalizing from it.

$$Entropy(S) = -\sum_{i=1}^{n} p(i) \log_2 p(i)$$

$$Gain(S, A) = Entropy(S) - \sum_{ve Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

J48 Pruned Tree, in Weka, resolves this issue by implementing pruning. In this method, some parts of the tree are cut off because they do not contribute much to the improvement of accuracy but instead add to the complexity. Pruning can be thought of as trimming unnecessary branches to strike a better balance between model simplicity and predictive performance. It's one handy solution to reduce overfitting, especially in datasets where patterns are vague or the class boundaries are noisy.

The Nearest Neighbors algorithm from Scikit-learn offers a completely different approach. Instead of building a global model, it relies on local patterns to classify a point based on its closest neighbors in the feature space. The class of a given data point is determined by the majority class among its *k*-nearest neighbors, in which the distance between points is computed by using some metric, such as Euclidean Distance:

$$d(x, y) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

While KNN is simple and effective for modeling local relationships, it does not scale well to very large datasets because of its high computational cost and may not perform well in high-dimensional spaces.

The algorithm we propose here combines the advantages of decision trees and KNN and gives a density-based decision tree that adapts to the data structure. Traditional decision trees split nodes recursively until all samples within a leaf come from the same class. This turns out to be problematic if some areas are sparse, as one might end up getting very specific splits that do not generalize well. So, in order to solve this problem, we implemented some density-based logic by incorporating KNN.

Here's how this works: at each node, the algorithm finds out the density of the data points in the region using:

$$\rho = \frac{k}{V}$$

where k is the number of data points within a given volume V. If the density falls below a certain value that is a predetermined threshold, then the tree will stop splitting and the node becomes a leaf. The intuition behind this is pretty straightforward — if there aren't enough data points to justify further splits, the tree stops. This approach avoids creating branches that are overly specific to a few outliers or noise in low-density regions.

On the other hand, it keeps on splitting in high-density regions to capture finer distinctions that enable it to model complex patterns in the data. The balance between splits and stopping ensures that the tree is not just after purity but is also taking a look at the spatial distribution of data. For instance, consider data of accidents that cluster in specific conditions—say, rainy nights on highways.

This is because our density-based approach allows the model to focus on these clusters without wasting time on sparse regions that don't add meaningful information.

This method combines density thresholds and KNN-inspired logic to produce a decision tree that dynamically adapts to the structure of the dataset. It yields a model that maintains the interpretability and speed of a decision tree while avoiding overfitting and dealing with complex and imbalanced datasets more effectively. This hybrid approach marries global decision-making with local pattern recognition and is quite suitable for real-world applications where data is noisy or not well-distributed.

5 Experiments

In order to assess the performance of our models, a number of relevant parameters for each algorithm were used. For the adaptive decision tree, the value of a maximum tree depth (maxDepth) is set to 10, a minimum density threshold (minDensity) is set to 2, and for the radius in the density estimation, the value is set to 14.0. For the process of choosing these parameters, they were tested with different values, and the best ones were then selected. These were tuned iteratively by creating a balance in the trade-off between capturing complex patterns and avoiding overfitting. The density threshold prevented the model from creating excessive splits in sparse regions while the depth limit controlled over-complexity in high-density areas.

Accuracy, precision, recall, the F1-score, and confusion matrices were used for the evaluation. Specifically, we used the macro-average precision and recall as it gives us a different metric from accuracy. Although it weighs the classes in the same sense, the micro-average precision and recall gives us no new information about the performance as it always equals the accuracy in a multi-class system. These metrics detail not only the general performance but also the performance related to how well each category has been correctly classified by the model.

6 Results

Model	Training Accuracy	Testing Accuracy
Control Decision Tree	99.5%	49.2%
Weka J48 Pruned Tree	74.0%	54.6%
Adaptive Decision Tree	52.0%	73.3%

Table 2: Accuracy Comparison of Models

Our adaptive decision tree outperformed both the control and Weka J48 models in generalization, achieving the highest testing accuracy of 73.3% by focusing on meaningful data patterns. In contrast, the control decision tree, while achieving a near-perfect training accuracy of 99.5%, struggled to generalize and yielded a much lower testing accuracy of 49.2%, indicating significant overfitting. The Weka J48 Pruned Tree showed some improvement, with a testing accuracy of 54.6%, but still failed to capture complex relationships in the data as effectively as our adaptive model.

	Predicted		
Actual	**Low**	**Moderate**	**High**
Low	333	0	1
Moderate	2	167	0
High	167	0	55

Table 3: Training Confusion Matrix for Control Model

	Predicted		
Actual	**Low**	**Moderate**	**High**
Low	92	34	18
Moderate	40	25	7
High	15	8	1

Table 4: Testing Confusion Matrix for Control Model

The control decision tree achieved a training accuracy of 99.5%, and as shown in the confusion matrix, which revealed near-perfect classification on the training set, we can see significant overfitting. However, when evaluated on the testing set, the performance dropped drastically, with an accuracy of only 49.2%, and we can see the poor generalization of our model through the testing confusion matrix.

	Predicted		
Actual	**Low**	**Moderate**	**High**
Low	309	25	0
Moderate	75	91	3
High	37	5	13

Table 5: Training Confusion Matrix for Weka J48 Pruned Tree

	Predicted		
Actual	**Low**	**Moderate**	**High**
Low	117	24	3
Moderate	55	14	3
High	16	8	0

Table 6: Testing Confusion Matrix for Weka J48 Pruned Tree

This tree struck a better balance than the control, achieving a training accuracy of 74% and a testing accuracy of 54.6%. While the model avoided severe overfitting, its performance on the test set revealed challenges in distinguishing between the *Moderate* and *High* classes due to overlapping feature distributions.

	Predicted		
Actual	**Low**	**Moderate**	**High**
Low	269	43	22
Moderate	139	18	12
High	47	5	3

Table 7: Training Confusion Matrix for Adaptive Decision Tree

	Predicted		
Actual	**Low**	**Moderate**	**High**
Low	131	6	7
Moderate	38	31	3
High	9	1	14

Table 8: Testing Confusion Matrix for Adaptive Decision Tree

Our proposed adaptive decision tree got a training accuracy of 52.0%, which appears lower than the other models but reflects its intentional design to avoid overfitting. The testing accuracy, however, improved by 20+%, getting to 73.3%.

Model	Precision	Recall	F1-Score
Control Decision Tree	0.345	0.343	0.344
Weka J48 Pruned Tree	0.309	0.336	0.322
Adaptive Decision Tree	0.712	0.641	0.675

Table 9: Precision, Recall, and F1-Score Comparison of Models for Testing Data

Our novel decision tree obtained a higher macro-averaged precision, recall, and F1-Score than the Control and Weka decision trees by around 0.4, demonstrating its ability to better handle the imbalance of classes in the decision tree. The results show us how the density-based stopping mechanism allowed the model to generalize a lot better to unseen data by focusing on high-density regions and avoiding splits in sparse areas.

7 Discussion

From these experiments, some important differences that show in the various models are how they balance training performance and generalization. The Control Decision Tree and Weka J48 Pruned Tree showed tendencies toward overfitting to the training data. For example, the control tree was able to fit

the training data with a very good accuracy of 99.5%, but the price paid was its testing accuracy dropping to as low as 49.2%. This is a real example of overfitting, wherein the model has memorized the training data instead of learning patterns that could generalize to unseen examples. Weka tree outperformed this by embedding pruning, which reduced unnecessary branches and slightly improved generalization. However, even with a testing accuracy of 54.6%, it was still not good enough to handle such overlapping features between *Moderate* and *High* accident severity.

Our adaptive decision tree took a very different approach, as will be obvious from the results. The main idea in our approach has been to forego the need for high training set accuracy and introduce an early stop to splits flailing in low-density regions; partitioning makes no sense there. This eventually brought us a training accuracy of 52.0%, which, though looking a bit low at the very beginning, was actually a very reasonable price for the benefit accompanying it: testing accuracy reached the top among all models, 73.3%. What happened here is that the model was biased more toward generalization rather than perfectly fitting to the training data, hence being better prepared for diverse real-world data.

One of the greatest contributors to the success was adding the KNN-inspired density calculation to this decision tree: at each node, a density calculation from the data point in the pertinent region using radius 14.0. If the density fell below a certain threshold, the model stopped splitting, meaning that there's not enough data here to make meaningful decisions. This approach ensured that the model avoided creating overly complex branches in sparse regions, which would have been prone to overfitting. For example, the imposed density threshold in regions of overlap between *Moderate* and *High* severities ensured that the model did broader splits which captured the overall structure rather than overfitting to the noise. This dynamic adjustment made the tree smarter as to where it should focus its efforts.

That said, our model isn't perfect. For instance, it fails to make some correct classifications when data is sparsely covered. For example, the *High* class had relatively few examples, and the model ended up not predicting that class for any cases. Then again, having a density threshold helped to reduce overfitting, but there were places where the tree could have generated finer splits of cases.

The best general performance is obtained with the adaptive decision tree, which does not have the disadvantages of classic trees and makes smart choices using local density information. It handled the unbalanced distribution of the dataset very well: thoroughly focusing on areas with meaningful patterns and skipping sparse areas that would have resulted in overfitting. Although there is room for improvement in handling outliers and rare cases, results show this approach will strike a good balance among simplicity, accuracy, and generalization.

8 Conclusion/Future Work

Our research sought to overcome one of the key challenges of decision trees: overfitting, which easily occurs in imbalanced datasets with overlapping features. Our adaptive decision tree model emerged as the top performer with a testing accuracy of 73.3%. Whereas the control and Weka J48 trees were biased toward high training accuracy - often at the cost of generalization - our model approached things differently. In fact, the KNN-inspired density-based stopping criteria avoided the splits of this tree when facing low-density areas, allowing it to focus on the most important data patterns.

One thing we may want to focus on is the sparse regions of the model, where the density threshold sometimes causes it to miss the subtlety in the pattern. One could look further into better ways of density estimation or refinement in setting the threshold value. The model could then be tested on larger and more diverse data to fine-tune the parameters, thus checking the scalability. Long term, we're excited about using this model in real-world applications, like accident severity prediction pipelines, to see how it performs under practical conditions. While this adaptive tree already showed impressive results, there's a lot of potential for it to evolve into an even more robust solution.

References

[1] I. Chaabane *et al.*, "Adapted Pruning Scheme for the Framework of Imbalanced Data-sets," *Procedia Computer Science*, vol. 112, Jan. 2017, pp. 1542–53. Available: https://doi.org/10.1016/j.procs.2017.08.060.

[2] S. M. Mazinani and K. Fathi, "Combining KNN and Decision Tree Algorithms to Improve Intrusion Detection System Performance," *International Journal of Machine Learning and Computing*, vol. 5, no. 6, Dec. 2015, pp. 476–79. Available: https://doi.org/10.18178/ijmlc.2015.5.6.556.

[3] I. Frias-Blanco *et al.*, "Online Adaptive Decision Trees Based on Concentration Inequalities," *Knowledge-Based Systems*, vol. 104, Apr. 2016, pp. 179–94. Available: https://doi.org/10.1016/j.knosys.2016.04.019.

[4] *View of Integrating Decision Tree and KNN Hybrid Algorithm Approach for Enhancing Agricultural Yield Prediction*. Available: https://cims-journal.net/index.php/CN/article/view/17/17.

[5] V. G. Costa and C. E. Pedreira, "Recent Advances in Decision Trees: An Updated Survey," *Artificial Intelligence Review*, vol. 56, no. 5, Oct. 2022, pp. 4765–800. Available: https://doi.org/10.1007/s10462-022-10275-5.

[6] Kuznetz, Den. *Traffic Accident Prediction*. Kaggle, 2024, https://www.kaggle.com/datasets/denkuznetz/traffic-accident-prediction

[7] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011.

[8] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). Available: https://doi.org/10.1038/s41586-020-2649-2

[9] The pandas development team. (2024). *pandas-dev/pandas: Pandas (v2.2.3)*. Zenodo. https://doi.org/10.5281/zenodo.13819579