Adaptive Branching: A Constraint Satisfaction Approach to Decision Tree Enhancement

Leah Zhang and Victoria Zhang

Jan. 26, 2025

Machine Learning 1 - Dr. Selma Yilmaz (Period 4)

Quarter 2 Project

ABSTRACT

Decision trees are a fundamental model within machine learning. However, it is well known that a major limitation of decision trees is their inability to handle overfitting. This project focuses on enhancing decision tree algorithms through the integration of dynamic constraint adjustments and optimized pruning strategies. Additionally, we test the two approaches of simulated annealing and differential equations for hyperparameter tuning of this modified decision tree, determining whether a stochastic or gradient-based approach is more effective for optimization. This approach aims to improve the adaptability of decision trees to real-time performance metrics, effectively balancing model complexity and interpretability while minimizing overfitting. Testing on a Kaggle dataset on credit risk analysis (Pore, 2024), we found that while the modified constrained optimization decision tree performed similarly to the baseline decision tree, simulated annealing improved overall accuracy. In comparison, the differential equation optimization approach, though having a higher accuracy than even the ensemble random forests model, failed to escape class imbalance, demonstrating the greater potential for the stochastic simulated annealing optimization method over the gradient-based differential equations method. This approach using constrained optimization in conjunction with a simulated annealing optimization metaheuristic demonstrates the potential for mimicking the higher accuracies and complexities of ensemble methods without the intense computational expense expected of ensembles for large datasets, as might be expected in complex real-world applications.

Introduction

Decision trees are a common classification algorithm prized for its simplicity and computational efficiency. However, especially in contrast to its ensemble learning method of random forests, decision trees struggle with overfitting. To improve the performance of decision trees without sacrificing computational efficiency as one would expect from random forests, we plan to investigate a new algorithm that can prevent overfitting for decision trees. This approach aims to revise decision tree algorithms to be both accurate and efficient when working with large-scale datasets.

These large-scale datasets are especially prevalent in finance, where reliable classification algorithms are needed specifically in credit risk analysis. In our paper specifically, we will utilize a dataset that uses loan applicant data, including age, income, and loan intent, to predict the likelihood of loan default.

Related Work

Relevant work on decision trees and their enhancements in literature can be categorized into three main areas: traditional decision tree algorithms, state-of-the-art ensemble methods, and constraint optimization techniques.

Traditional decision tree algorithms such as Classification and Regression Trees (CART) introduced by Breiman et al. (1986) and Quinlan's ID3 and C4.5 (1986, 1993) are regarded as the foundation of revised decision tree algorithms. Of these methods for building decision trees, the greedy method, in which a tree is built upon each node by considering the immediate information gain or other selected attributes, is popular. However, though these approaches offer efficient performance, particularly with smaller datasets, these traditional approaches face significant limitations: they carry a proclivity to overfit data, especially as trees grow unnecessarily complex, capturing noise rather than true underlying data. Moreover, these algorithms typically rely on static thresholds for making splits and pruning, hindering adaptability when faced with varying data distributions.

To combat overfitting, pruning techniques are used to remove branches that do not contribute meaningfully to the model's predictive power. However, although pruning methods are relatively straightforward to implement and interpret, their static nature may result in suboptimal tree structures, as they do not account for dynamically changing variables and performance data with each iteration of the model. Therefore, current state-of-the-art approaches aim to take these limitations into consideration. These approaches include ensemble methods such as random forests and gradient boosting machines

(XGBoost, LightBGM, CatBoost), which have become popular due to their superior performance over decision trees in various applications. These methods use multiple decision trees to improve robustness and reduce overfitting by averaging predictions or utilizing boosting techniques. However, these ensemble methods often sacrifice interpretability, one of the key strengths of traditional decision trees.

To build upon adaptability, another approach uses **constraint optimization techniques** to transform classification tasks into Constraint Satisfaction Problems (CSPs). This approach was used by Pendharkar (2005), which optimized ANN classification by adjusting constraints throughout the learning process through a constraint-based search method. Verhaeghe et. al. (2020) extends this application of constraint programming to construct optimal decision trees, incorporating constraints on depth and balance during the tree-building process. This approach facilitated the efficient pruning of unpromising branches early, improving computational efficiency. Generalizing these approaches to dynamic programming, Emir Demirović et al. (2022) introduces how optimal decision trees can be learned from a global perspective by leveraging dynamic programming principles combined with search strategies. Rather than a greedy method iteratively seeking for the local optimum at each node, this algorithm optimizes the entire tree globally by systematically evaluating potential splits and their impacts on the final model, greatly improving efficiency. All these dynamic methods' strength lies in their flexibility to adapt to changing problems and data characteristics with each iteration of the tree-building, leading to improved decision quality. Yet the inherent complexity of such dynamic systems also entails higher computational overhead, making them less suitable for general real-time applications.

Project Goal

The methodologies above share a common goal of optimizing decision trees, yet they differ in how they approach model building and constraint integration. This project distinguishes itself by not only incorporating dynamic adjustments based on real-time performance metrics, but also integrating a robust pruning strategy informed by the adaptability of constraints. By building upon existing dynamic programming strategies while also emphasizing real-time adaptability, this project aims to construct decision trees without sacrificing either accuracy or efficiency, while also avoiding the lack of interpretability found in state-of-the-art ensemble methods.

Dataset and Preprocessing

Our dataset is on credit risk analysis (Pore, 2024). It is a dataset that provides essential information about loan applicants. We will be classifying the likelihood of the loan defaulting, when a

borrower stops making payments on a loan or credit card, based on factors such as age, income, and home ownership status. We decided to use this dataset because of the large number of instances, ~32,800, and its application toward today's financial landscape. These two details will help us best demonstrate the benefits of our dynamic method for optimizing the hyperparameters.

Attribute	**Description**
ID	Unique identifier for each loan applicant.
Age	Age of the loan applicant.
Income	Income of the loan applicant.
Home	Home ownership status (Own, Mortgage, Rent).
Emp_Length	Employment length in years.
Intent	Purpose of the loan (e.g., education, home improvement).
Amount	Loan amount applied for.
Rate	Interest rate on the loan.
Status	Loan approval status (Fully Paid, Charged Off, Current).
Percent_Income	Loan amount as a percentage of income.
Cred_Length	Length of the applicant's credit history.

Table 1: Description of Attributes

Preprocessing of the data can be described as follows. An attribute expected to be irrelevant to classification, the 'Id' variable, was removed. Missing values found in 'Emp_length' and 'Rate' were replaced with the median of existing data. Categorical variables, such as 'Home' and 'Intent' and including the class variable of 'Default' were numericized via one-hot encoding. Finally, numerical variables of 'Income,' 'Amount,' and 'Percent_income' were normalized through MinMaxScaler().

Following the above steps, we used stratifying sampling to split the dataset into training and testing sets with a 0.2 test size. After this, we needed to address the class imbalance. Two values are possible for the class 'Default': true and false. There are around 26.8k values for false, 82% of the total, and 5745 values for true, 18% of the total. To address this we decided to use the Synthetic Minority Oversampling Technique (SMOTE). This method takes existing examples in the minority class and creates synthetic data. Using this technique on the training set, which initially had 26063 data points with 21468 false and 4596 true, the distribution was now 21468 for both false and true. This is saved as the CSV 'training_set_resampled'.

Methods

To see how various methods could address class imbalance and accuracy, we tested five separate approaches:

1. Model 1: **A baseline decision tree mode**l with no modifications
2. Model 2: **A random forest model** a pre-existing solution to overfitting to use as comparison
3. Model 3: **A novel modified constraint optimization approach** that introduces constraint optimization and dynamic pruning approaches
4. Model 4: **A novel simulated annealing-enhanced modified decision tree**, which introduces simulated annealing (SA) to optimize the hyperparameters introduced by Model 3.
5. Model 5: **A novel differential equation-optimized modified decision tree**, which, similar to the above approach, uses differential equations to optimize the hyperparameters introduced by Model 3.

Model 1: The Baseline Model

The baseline decision tree model is a standard implementation of a decision tree using the scikit-learn library. We chose Gini impurity over information gain to avoid computationally intensive logarithm calculations. For n total classes and p_i describing the probability of samples belonging to class i at a given node, the Gini impurity of dataset D can be denoted as follows;

$$Gini((D) = 1 - \sum_{i=1}^{n} (p_i)^2$$

Equation 1. Formula for Gini impurity of a dataset D

The baseline decision tree was built with max_depth=4, a parameter that would remain constant in any following experiments. We used the evaluation metrics of accuracy, F1-score, precision, recall, and ROC-AUC to evaluate performance, as well as a confusion matrix visualization. These evaluation metrics would continue to be used for all following experiments.

Model 2: Random Forest/Ensemble Learning

Before we move on with our changes to existing approaches, we would like to test out our model against another benchmark: Ensemble Learning. Ensemble learning is a preexisting solution that addresses overfitting by using multiple decision trees instead of one. We will be using the 'RandomForestClassifier' model from Scikit-learn. We decided to use 1000 trees and random state 42.

Model 3: The Modified Constraint Optimization Approach:

A general overview of the initial modifications our model makes on the classical decision tree can be described as follows:

Step	Classical Approach	Constraint-Based Approach
Feature Selection	Gini or entropy-based splitting	Penalized gain with dynamic thresholds
Pruning	Cost-complexity pruning	Post-pruning using gain penalties
Computational Complexity	Greedy exploration of all branches	Prioritized search with reduced search space

Table 2: Overview of differences in approaches

1) Feature Selection with Dynamic Information Gain

To implement a dynamic feature selection process, we replace the default information gain or Gini impurity selection in DecisionTreeClassifier with a custom calculation that includes a penalty for small gains. Specifically, this new custom gain function incorporates penalties for small information gains using a dynamic threshold:

$$G_{adjusted} = G - max(0, P * (T - G)), \quad (2)$$

where G is the standard gain or impurity calculation, P is a penalty factor, and T is a minimum gain threshold. This computation would then be applied in a custom loop to evaluate and choose features before splitting each node. This dynamic adjustment discourages splits that do not significantly improve purity, particularly in scenarios with imbalance classes such as in this dataset.

2) Constraint-Based Pruning

We conducted pruning by traversing the tree node-by-node, removing branches when the gain constraint is violated (the adjusted gain falls below zero).

Model 4: Enhancing the Modified Tree with Simulated Annealing:

Simulated annealing is a common optimization metaheuristic inspired by metallurgical annealing, where a material is heated and slowly cooled to remove defects. Translating to the simulated annealing algorithm, the “heat” describes the randomness in the search during the optimization process, which decreases over time to refine the solution. SA is a probabilistic algorithm whose strength lies in its ability to escape local minima, allowing it to find near-global optima in large and complex search spaces typical of real-world issues. The general process of SA can be described as follows:

1. Initialize an initial solution and calculate its fitness (objective value)
2. Generate new solutions by perturbing the current solution slightly to explore the search space.
3. Evaluate the new solutions by determining fitness. If the new solution is better, accept it as the current solution.
4. If the new solution is worse, accept it with probability: $P(accept) = exp(\frac{\Delta fitness}{temp.})$, where $\Delta fitness$ refers to the difference in fitness between this new solution and the current solution. This step of **probabilistic acceptance** is crucial to allow the algorithm to escape local optima early in the search.
5. Update temperature using a set cooling rate: $T_{new} = T_{old} * cooling\ rate$. As the temperature decreases, the algorithm becomes more selective in accepting worse solutions
6. Repeat steps 2-5 (generating and evaluating new solutions) until a stopping criterion is met (max. number of iterations or a sufficiently low temperature is reached)
7. Return the best solution found after the final iteration.

In the context of our decision tree implementation, SA is used to optimize the hyperparameters P (penalty factor) and T (minimum gain threshold) that we introduced in the previous constrained optimization approach. Specifically, the SA algorithm used in this enhanced approach begins with an initial solution of $P = 0.1$, $T = 0.01$. Fitness for this solution and subsequently-generated solutions was calculated with a function that evaluated the trade-off between accuracy and complexity via a Lagrangian objective:

$$Fitness = Accuracy - \lambda * Tree\ Complexity \quad (3)$$

where accuracy is evaluated through prediction quality on the test set, tree complexity is proportional to the number of nodes in the tree, and $\lambda = 0.01$, an arbitrarily set ratio of accuracy and complexity. The use of this Lagrangian objective is intended to maintain the balance between maximizing accuracy while minimizing tree complexity (reducing overfitting), which was the original goal of the modified decision tree algorithm The result of the SA algorithm in this context identifies the optimal values of P and T to use in the above-modified decision tree approach, which are not trivial to tune manually due to their complex interactions with tree performance.

Model 5: Alternative Method Using Differential Equation Optimization:

A differential equation (DiffEq) is a mathematical equation that relates a function to its derivatives. These derivatives describe how a quantity changes over time-based on its current value, making DiffEq a strong tool to model dynamic systems that change continuously. In optimization problems, DiffEq can be used to model the evolution of parameters or states under a set of rules derived from the problem's objective function and constraints. While SA tests the ability of stochastic models to handle complexity and randomness in tree-based models, differential equations can instead examine whether gradient-based fine-tuning is more effective in improving model performance. Similar to our

earlier implementation of SA, we test whether DiffEq is an effective method to optimize the modified decision tree hyperparameters P, T.

Similar to the fitness function used for simulated annealing, we can represent the system in terms of its cost function, in which we expect the parameters P and T to evolve over time according to their gradients. This cost function can be represented as:

$$J(P, T) = Accuracy - \lambda * Number\ of\ Nodes. \quad (4)$$

The parameter updates are gradient-based, governed by an ODE:

$$\frac{dP}{dt} = -\frac{\partial J}{\partial P}, \quad \frac{dT}{dt} = -\frac{\partial J}{\partial T} \quad (5)$$

These equations describe how P and T should evolve to minimize the cost function J. The gradients can further be approximated using finite differences:

$$\frac{\partial J}{\partial P} \approx \frac{J(P+\epsilon, T)-J(P,T)}{\epsilon}, \frac{\partial J}{\partial T} \approx \frac{J(P,T+\epsilon)-J(P,T)}{\epsilon} \text{ for some small step size } \epsilon > 0. \quad (6)$$

In Python, these equations are solved using the numerical scipy solver *solve_ivp*, which updates P and T continuously based on their gradients. The optimization stops when either the system reaches the final time point or when the gradients approach zero, indicating convergence. After optimization, the best parameters P and T are used to train and prune the modified decision tree described earlier.

Results and Discussion

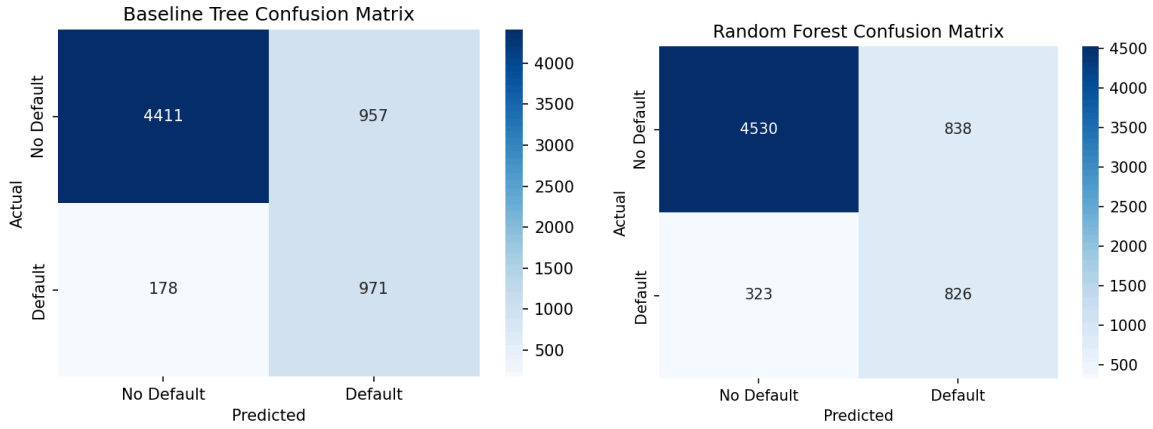

Figure 1: The two models that were used as a baseline.

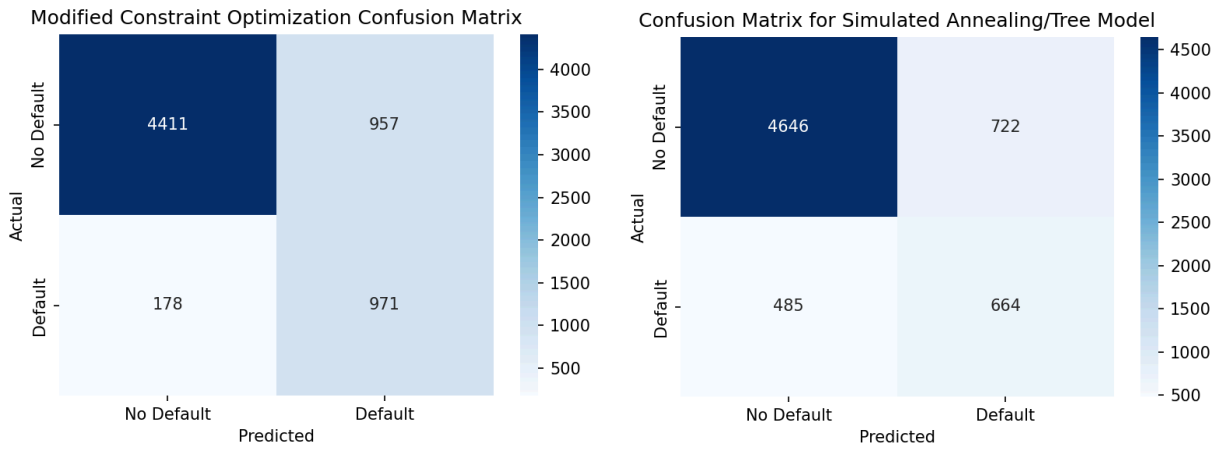

Figure 2: The two models that used constraint optimization

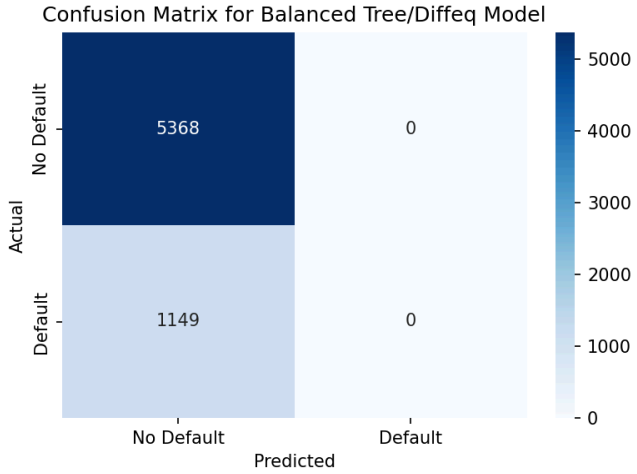

Figure 3: Using the Diffeq Model

	Accuracy	F1 Score	Precision	Recall	ROC AUC
Baseline Tree	0.80	0.49	0.45	0.54	0.70
Random Forest	0.82	0.59	0.50	0.72	0.88
Constraint Optimization	0.80	0.50	0.45	0.55	0.68
Simulated Annealing	0.81	0.52	0.48	0.58	0.78
Differential Equation Optimization	0.8237	0	0	0	0

Table 3: Table of Results

The results shown from Figures 2, 3, 4 and Table 3 all have around 81% accuracy. Even if we don't account for the minority class, the accuracy is still around 81 percent. There are several possibilities for why this is, but we believe this occurs because the dataset is too straightforward. Decision tree models are prone to overfitting when too many branches are created. This occurs when there are either too few data points or a highly noisy dataset. Since the data is from Kaggle where polished datasets are usually uploaded, this could not occur. Thus, our approach was unable to showcase its advantage to the baseline model. This is also why the Random Forest model, which is a pre existing solution to overfitting, also has an accuracy around 0.82.

One of the bigger questions is why does differential equation optimization (Model 5) always select False to the default? Doing this does give it the highest accuracy, but this is only because of the class imbalance. The deterministic nature of the DiffEq method makes it such that the path of optimization is always directed towards the gradient of the cost. If these gradients point towards a local optima that heavily favors the majority class, as is not unlikely due to the cost metric using f1-score, the DiffEq optimization is less likely to escape this local optima. Unlike the simulated annealing approach, which does have an in-built mechanism to avoid being trapped in local optima, this key weakness of the DiffEq gradients could have caused the heavy imbalance in its classification, leading to its precision, Recall, AUC, and F1 Score all being 0.

Conclusion

Overall, we have shown that our method for dynamic feature selection, as shown in Models 3-5, is possible and applicable to decision trees. Although our results are relatively similar to the baseline decision tree model (Model 1), our model performs slightly better in all metrics, especially with enhanced parameter optimization as shown in Models 4-5. In addition, our results accomplished our objective of creating a dynamic feature selection algorithm that reduces overfitting. In the future, we hope to use a wider variety of datasets to find out when our model will best cover the gaps left by the base decision tree. Either we will explore a smaller dataset, to which decision trees are prone to overfitting with, or a more complex dataset based on government data.

References

Emir Demirović, Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Kotagiri Ramamohanarao, & Stuckey, P. J. (2022). MurTree: Optimal Decision Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, *23*(26), 1–47. https://www.jmlr.org/papers/v23/20-520.html

Fajemisin, A. O., Maragno, D., & den Hertog, D. (2024). Optimization with constraint learning: A framework and survey. *European Journal of Operational Research*, *314*(1), 1–14. https://doi.org/10.1016/j.ejor.2023.04.041

Pacheco, J., Alfaro, E., Casado, S., Matías Gámez, & García, N. (2012). A GRASP method for building classification trees. *Expert Systems with Applications*, *39*(3), 3241–3248. https://doi.org/10.1016/j.eswa.2011.09.011

Pendharkar, P. C. (2006). A data mining-constraint satisfaction optimization problem for cost effective classification. *Computers & Operations Research*, *33*(11), 3124–3135. https://doi.org/10.1016/j.cor.2005.01.023

Pore, N. (2024). *Credit_Risk_Analysis*. Www.kaggle.com. https://www.kaggle.com/datasets/nanditapore/credit-risk-analysis

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/bf00116251

Salzberg, S. L. (1994). C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Machine Learning*, *16*(3), 235–240. https://doi.org/10.1007/bf00993309

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., & Schaus, P. (2020). Learning optimal decision trees using constraint programming. *Constraints*, *25*(3-4), 226–250. https://doi.org/10.1007/s10601-020-09312-3