Decision Tree Learning
Algorithms for Robust Prediction and Policy Optimization

Vos, D.A.

DOI
10.4233/uuid:5b430119-8ad4-4fae-8ff8-b31c70c1090d

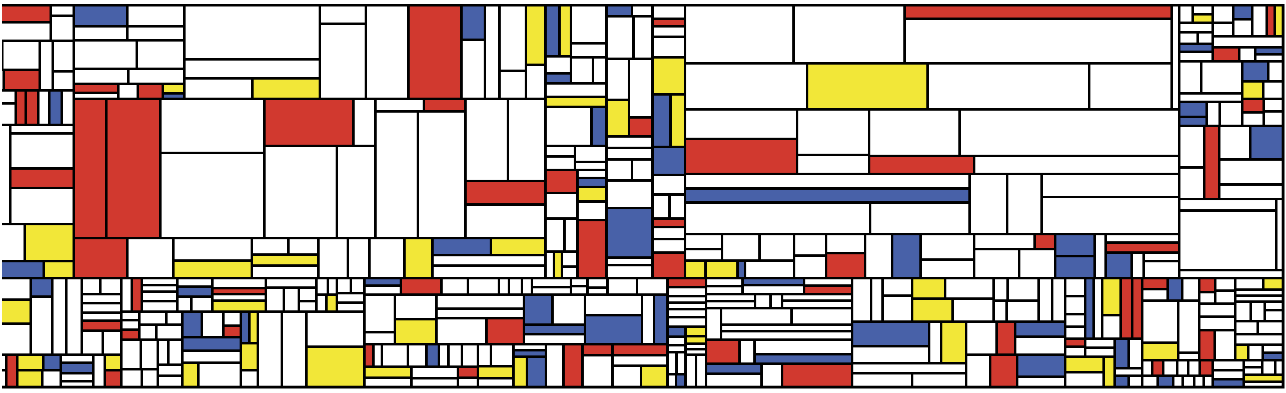
Publication date
2025

Document Version
Final published version

Citation (APA)
Vos, D. A. (2025). *Decision Tree Learning: Algorithms for Robust Prediction and Policy Optimization*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:5b430119-8ad4-4fae-8ff8-b31c70c1090d

Decision Tree Learning

Algorithms for Robust Prediction and Policy Optimization



Daniël Vos

Decision Tree Learning

Algorithms for Robust Prediction and Policy Optimization

Decision Tree Learning

Algorithms for Robust Prediction and Policy Optimization

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
maandag 6 januari 2025 om 17:30 uur

door

Daniël Alexander VOS

Master of Science in Computer Science,
Technische Universiteit Delft, Nederland,
geboren te Delft, Nederland.

Dit proefschrift is goedgekeurd door de promotoren.

The work in the thesis has been carried out under the auspices of The Netherlands Research School for Information and Knowledge Systems (SIKS).

Keywords: decision trees, machine learning, ensembles, adversarial examples, robust optimization, reinforcement learning, control

Printed by: Proefschriftspecialist

ISBN: 978-94-6384-716-2

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

Contents

Summary

We increasingly encounter artificial intelligence-based technology in our daily lives, from smart home devices to self-driving cars to invisible systems running on our internet. Many artificial intelligence techniques use machine learning, algorithms that learn to predict or act based on collected data. Unfortunately, the most popular machine learning techniques, such as neural networks and ensembles, are so complex that humans cannot understand how they make predictions. Without understanding the prediction process, it is difficult to trust the model. Therefore, in this dissertation, we work on algorithms that learn models that are understandable to humans. The type of model we consider is a decision tree, a flowchart-like model that can easily be visualized so humans can understand it. These models ask a series of questions about an input and use the answers to derive a prediction.

Decision trees were popularized in the 1980s and extensively studied, but there is still room for improvement. The most popular algorithms for learning decision trees are fast but do not necessarily lead to the best performance. They are not robust, meaning tiny changes in the data can negatively influence the quality of their predictions. Also, the existing algorithms cannot be directly applied to problems where multiple sequential predictions have to be made. Therefore, this dissertation studies several techniques for learning decision trees for robustness and sequential decision making problems.

In Part I of the dissertation, we consider the problem of optimizing decision trees to make good predictions while being robust to small changes in the data. In Chapter 4, we tackle the problem of learning good decision trees quickly in this setting. We improved the runtime of an existing algorithm by speeding up one of the key operations. In Chapter 5, we solve the problem of finding the best possible robust decision tree. The idea is to formulate the problem as an Integer-Linear Program, a special mathematical problem that can be solved with highly optimized algorithms. In Chapter 6, we propose a method that allows learning of models that are more flexible in terms of robustness, i.e., by allowing data changes in different shapes. To create an efficient algorithm, we optimize only the model's predictions, not the model's question part. Finally, in Chapter 7, we use techniques for improving data privacy to enable robustness against another kind of data change: someone adding or removing data.

Part II of this dissertation is about sequential decision making problems. In these settings, we control a device or agent that tries to achieve some goal in a potentially uncertain environment. Sequential decision making problems are significantly different from the supervised learning problems considered in Part I since the data is not pre-collected. This class of problems encompasses many real-life problems; one of the simplest of those could be a thermostat that measures the temperature in a room and needs to decide whether to turn a heater on or off constantly. Highly complex problems such as self-driving cars can be modeled similarly. We aim to find a controller represented by a simple decision tree for such problems. Such a controller is called a policy, and by representing it with a small decision tree, humans can understand it. In Chapter 9, we assume that we have a perfect mathematical description of the problem and use it to find the best possible decision tree via Integer Programming techniques. Later in Chapter 10, we assume that we can only interact with the environment and do not have a mathematical description of the problem. In this setting, we find good policies by iteratively updating the tree to achieve better scores using gradient information.

In our research, we have developed various algorithms for learning decision trees in settings that are hard to optimize with existing methods: robust predictions and sequential decision making. We hope that our work on decision tree learning for these settings allows human-understandable machine learning to be used in more real applications in the future. By improving model understanding and robustness, we aim to enable machine learning systems that humans can trust.

Samenvatting

In het dagelijks leven komen we steeds vaker in aanraking met technologie die gebruik maakt van kunstmatige intelligentie, van smart home apparaten, tot zelfrijdende auto's tot de onzichtbare systemen die op ons internet draaien. Veel technieken voor kunstmatige intelligentie maken gebruik van machine learning, algoritmes die leren voorspellen of keuzes maken aan de hand van data. Helaas zijn de meest gebruikte technieken voor machine learning, neural networks en ensembles, zo complex dat mensen niet kunnen begrijpen hoe de modellen voorspellingen maken. Zonder dat we de voorspellingen kunnen begrijpen is het moeilijk om de modellen te vertrouwen. In dit proefschrift proberen we daarom algoritmes te ontwikkelen die modellen kunnen leren die voor mensen te begrijpen zijn. Het type model waar we aan werken is een decision tree, een model dat lijkt op een stroomdiagram en makkelijk te visualiseren is zodat mensen hem kunnen begrijpen. Decision trees stellen een aantal vragen over de invoer en gebruiken de antwoorden daarop om uiteindelijk tot een voorspelling te komen.

Decision trees zijn populaire modellen sinds de jaren 80 en uitgebreid onderzocht, maar er is nog steeds ruimte voor verbetering. De meeste algoritmes om decision trees te leren zijn snel, maar vinden niet altijd de beste oplossing. Ze zijn niet robuust, waardoor voorspellingen kunnen veranderen door een miniscule aanpassing in de data. Ook zijn de bestaande methodes niet direct toepasbaar voor problemen waar meerdere afhankelijke voorspellingen achter elkaar moeten worden gemaakt.

In Deel I van het proefschrift bestuderen we het probleem van decision trees leren die goede robuuste voorspellingen maken en dus goed omgaan met kleine aanpassingen. In Hoofdstuk 4 werken we aan technieken om snel goede decision trees te vinden. We versnellen een bestaand algoritme door een van de belangrijke operaties veel efficiënter uit te voeren. In Hoofdstuk 5 werken we aan algoritmes die de best mogelijke robuuste decision tree vinden. Hiervoor formuleren we het probleem als een Integer-Linear Program, een speciaal soort wiskundig probleem dat opgelost kan worden met slimme algoritmes. In Hoofdstuk 6 presenteren we een methode die flexibeler is in termen van robuustheid, zo kunnen we bijvoorbeeld decision trees optimaliseren die robuust zijn voor aanpassingen in verschillende vormen. Om daar een efficiënt algoritme voor te ontwikkelen optimaliseren we de voorspellingen van de decision tree en niet de vragen. In Hoofdstuk 7 gebruiken we technieken voor het verbeteren van data privacy om robuustheid te realiseren voor een andere manier van aanpassen: als iemand data verwijdert of toevoegt.

Deel II van dit proefschift gaat over sequential decision making problemen. Bij dit soort problemen besturen we een apparaat of 'agent' die probeert een bepaald doel te bereiken in een onzekere omgeving. Dit soort problemen zijn anders dan de problemen uit Deel I omdat niet alle data van tevoren verzameld is. Sequential decision making problemen omvatten veel problemen die we ook in het dagelijks leven tegenkomen. Een van de simpelste varianten is bijvoorbeeld een thermostaat die de temperatuur van een kamer meet en continu moet beslissen of de verwarming aan of uit moet worden gezet. Maar

veel complexere problemen zoals zelfrijdende auto's kunnen zo ook worden gemodeleerd. Ons doel is om een controller in de vorm van een decision tree te vinden voor zulk soort problemen die de beslissingen kan maken. Zo'n decision tree wordt ook wel een policy genoemd, en door hem klein (en dus simpel) te houden, kunnen mensen hem goed begrijpen. In Hoofdstuk 9 nemen we aan dat we een volledige wiskundige beschrijving van het probleem hebben, en gebruiken die beschrijving om de best mogelijke decision tree te vinden met Integer Programming technieken. Later in Hoofdstuk 10 nemen we aan dat we alleen kunnen leren via interacties met de omgeving en zo proberen we om zonder wiskundige beschrijving van het probleem toch goede decision trees te optimaliseren. In deze situatie vinden we goede policies door steeds kleine aanpassingen te doen aan de decision tree die de score verhoogt aan de hand van de gradiënt.

In ons onderzoek hebben we verschillende algoritmes ontwikkeld voor het leren van decision trees voor taken die lastig te optimaliseren zijn met de bestaande methoden: robuuste voorspellingen en sequential decision making. We hopen dat ons werk meer toepassingen van begrijpelijke machine learning methoden mogelijk maakt in het dagelijks leven. Door het verbeteren van de begrijpelijkheid van modellen en hun robuustheid hopen we het mogelijk te maken om kunstmatige intelligentie systemen te ontwikkelen die mensen kunnen vertrouwen.

Acknowledgments

It is widely accepted that the PhD trajectory feels like a rollercoaster, and my PhD was no exception. From being stuck working and living in a single room during the pandemic to traveling all over the world to present at conferences, the working situation has changed a lot. However, one thing that has remained consistent is the constant support from the people around me.

First, I would like to thank my promoters for supervising me throughout my academic journey. When I met Sicco in his office in November 2019 to start my Master's thesis with him, I never imagined he would be my supervisor for the next 5 years. However, I am so happy this happened because his sympathetic guidance has made the process much easier, and I have learned so much from him over the years. I hope that in the future, we can continue our passionate conversations about machine learning, optimization, and interpretability, and maybe one day, the recurrent decision tree will actually exist. I also want to thank Inald for his supervision and for taking the time for our many meetings. I admire his ability to see through the details and ask the questions I never thought about (I also hope this does not foreshadow the type of questions he will ask during my defense).

I was very lucky to be born in a city with an excellent university, meaning I could study while living close to my family. It was a pleasure studying at the same university as my twin brother Jelle and my sister Evelien, and it allowed us to stay in close contact during this time. I trust them blindly and therefore there was no doubt asking them to be my paranymphs. I want to thank my parents for never questioning my choice of pursuing a PhD and supporting me throughout this endeavor. My grandparents, aunts, uncles, and cousins have also been great motivators. Although it is too late for him to read this now, I want to thank my grandfather for inspiring me. He was still learning about science at a late age, and I had hoped to introduce him to machine learning and optimization with this thesis. This goal has motivated me to work hard in the hopes that I could finish my thesis in time for him to see it.

The pandemic was a strange time to start the PhD. We were all working from home at the start, so it wasn't easy to meet my new colleagues. However, this challenge was no match for Sicco's other students, Azqa and Clinton, with whom I quickly became close friends. From silly online board games to unfortunate eggcidents, hysterical laughter was guaranteed every time we met. Their friendship and scientific collaborations have made the PhD many times easier. Later in my PhD, a new colleague joined our group: Simon. Clearly, he quickly integrated into our little community because, within a year, we were at Simon and Chengyao's wedding! It was an honor that they trusted me to be one of their witnesses, and I wish them lots of love in the future. I also want to thank Sicco's PhD students and PostDocs Robert, Tom, (other) Daniël, Ligia, Luca, and Chris for many good memories, and the great master students Maria, Cas, Wessel, and Bram, who I got to help with their research.

Something that made my PhD quite unique is that I got to be a part of not just one but two research groups at TU Delft. While I was initially worried about this development, I am now grateful for the wonderful people it has allowed me to meet in both groups. I completed my Master's thesis and the first half of my PhD at the cyber security research group, and although I moved groups, I have never truly let go of my cyber security background. I want to thank my colleagues there for many pleasant connections and for supporting the Capture-The-Flag (CTF) team. I am sure we will stay in touch.

During the first year of my PhD, I started the TU Delft CTF Team, a place for TU Delft students to learn about and participate in hacking competitions. We started with 4 people solving puzzles in a meeting room, and over the years, our team has grown to enormous proportions. We now have a community of hundreds of hackers, organize competitions with 200 participants, and have topped Dutch university rankings for multiple years. I am incredibly proud of the students who have dedicated time to making this team a success, and I want to thank them for inspiring me tremendously with their talent and ambition. The depth of the knowledge and skills of these young hackers is truly remarkable.

In the second half of my PhD, we moved to the Algorithmics research group. Although I hardly knew anyone there, the group quickly made me feel at home. I was surprised by how hard-work goes hand in hand with such a cheerful social culture, and it is a pleasure to come to the office every day. Although the group knows that I am one of the most frequent visitors of the coffee machine, I must admit it might not be the coffee's fault, but the fact that there is always an opportunity for a pleasant conversation. I feel like I have a connection with every student, PostDoc, and professor in the group, and I am very grateful for everyone being so open to this. I want to mention Koos, Anna, Emir, and Mathijs specifically for collaborating with me on decision tree learning.

The last year of my PhD started with a research visit on the other side of the world: USC in Los Angeles! I want to thank Phebe Vayanos for her supervision and both Sicco and Phebe for making this visit possible. I am also grateful that I got to meet all the PhD students at the Center for Artificial Intelligence in Society, who made my stay so much more fun. I want to say extra thanks to Bill, Nathan, and Caroline, who showed me around, took me bouldering, brought me to a true American college rager, and surprised me with some of the fanciest donuts I have ever seen to celebrate my birthday. Due to an ongoing emergency back at home, their kindness meant the world to me.

Lastly, I want to mention the support of my friends outside of the research groups. Meeting my friends from my earlier studies, Proteus, X, and high school, always gave me a boost. For a long time, Niv dealt with my complaints about the PhD and has always encouraged me to keep going. I hope she feels the same level of support in her own PhD journey. I also want to thank many of the colleagues and cool kids I met at summer schools, SIKS courses, for the adventures we had at these trips.

You might have noticed that I included few names in this section. I want these acknowledgements to not just be a list of names, as this would not do justice to my countless experiences with everyone. Rather, I want to convey how much these groups of people have meant to me and how they helped me get through the PhD, and I hope you will recognize yourself as being part of these groups. If in the future, the people around me will be anything like the people in the past, I might just stay in academia forever!

Daniël

1

General Introduction

Machine learning techniques are increasingly deployed in products that we encounter in our daily lives. For example, chatbots such as ChatGPT are used by over a hundred million weekly users [1] and use machine learning models to generate natural language. However, it is hard to trust the predictions of many machine learning models. ChatGPT is notorious for telling convincing lies [2], and since the model behind ChatGPT is too complex to be human-interpreted, it is challenging to determine when the model will lie. Moreover, neural networks are overly sensitive to tiny changes in the input, making them brittle, and can reveal private data they were trained on. This results in models that make mispredictions when single pixels in an image are changed. In contexts where we need to rely on the predictions of a machine learning model, we need the model to be human-understandable. A promising type of model for this is the decision tree [3].

Decision trees are simple machine learning models. These models can be visualized by flow-chart-like diagrams where one follows a path through the diagram based on a series of questions and ends in a leaf node that holds a prediction. When limiting the size of decision trees, they are intuitive and can be readily understood by humans. Therefore they can be used to solve the problem of transparency in machine learning, but they still have their limitations. Just like neural networks, they are brittle in the presence of tiny changes in the data. Also, they cannot be applied to sequential decision-making problems in which multiple co-dependent predictions have to be made. We want to find decision trees that make good predictions, are robust to small changes and can be applied in sequential decision making.

Unfortunately, optimizing decision trees for arbitrary objectives is a difficult task. The most popular techniques in machine learning rely on gradient descent, an optimization method that makes small incremental updates to the model. Decision trees do not allow for this approach since the true/false questions they consist of do not smoothly change and therefore cannot be differentiated. Instead, decision trees require specialized optimization methods. While there is a rich literature on decision tree optimization techniques for classical classification and regression tasks, these methods do not extend directly to other settings. Therefore, in this dissertation, we consider the problem of optimizing decision trees for hard-to-optimize tasks that are currently not handled well by existing methods. Specifically, we study optimizing decision trees for robustness and sequential decision making problems. Although the problems are different, we can use similar algorithmic ideas to optimize decision trees for them.

1.1 Optimizing Decision Trees

Decision trees are promising models for their interpretability when limited in size. We briefly introduce decision tree models and the problem settings that we apply them to: robust learning and sequential decision making.

1.1.1 Decision Tree Models

Decision trees, such as the one visualized in Figure 1.1 (left), are hierarchical models that combine a series of tests on information from an input to determine its prediction. When limiting a decision tree's size, humans can easily interpret it. In recent years, this property has motivated more research into decision trees in response to the increasing popularity

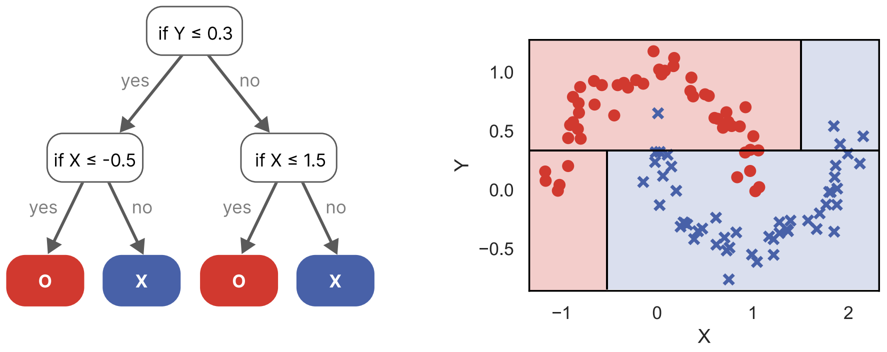


Figure 1.1: (left) A decision tree of depth 2. By following the path from the root node to a leaf, we can find the prediction for an instance. For example, for an instance with $X = 2.0, Y = 0.0$, we follow the path left of the root node since $0.0 \le 0.3$ and right at the second node since $2.0 \not\le -0.5$, the leaf that we end up in predicts the class 'blue cross.' (right) The training dataset for this tree is visualized by circles and crosses, and the box-shaped regions the decision tree leaves cover. The background color of each box represents the prediction of the associated leaf.

of extremely complex models such as neural networks. An added advantage of decision trees over other interpretable methods, such as linear models, is that they can model non-linear relationships in an interpretable way. For example, a linear model cannot accurately predict the dataset in Figure 1.1 (right), but a decision tree can. In this figure, the decision tree correctly separates most of the red and blue points by splitting up the space in only 4 regions.

A single decision tree (see Figure 1.1) is usually represented by a binary tree of decision nodes that contain tests on a feature and leaf nodes that hold a prediction. The root node of a decision tree is the node that starts the prediction process and is usually placed at the top of the diagram. To make a prediction, we compare the feature values of a data instance to the root node and follow the path to the leaf that satisfies all decision nodes. The final prediction is simply the value indicated by the leaf.

Decision trees are usually limited in size by either limiting the depth of the tree (the length of the longest decision path), limiting the number of nodes (or, equivalently, the number of leaves), or both. While it is possible to create decision trees with more than 2 decisions at each node, we will limit ourselves to binary trees in this work. This is because they naturally encode true/false relations that are easy for humans to interpret and are as powerful as trees with more than 2 decisions per node.

Decision Node Predicates Binary decision trees model true/false decisions where an instance follows either the left or right path at each decision node. A common predicate to use for these decisions in the case of numerical data is the less-than-or-equal predicate ($\le$) which compares the value of an instance's feature to a fixed threshold value. However, in real-world applications, datasets often contain a mix of numerical feature data such as 'weight' or 'size' and categorical features such as 'student/teacher' or 'language,' and these categorical features benefit from using different predicates. Some of the algorithms in this work use specialized predicates for such cases while other algorithms rely on the user to encode such features with numerical values.

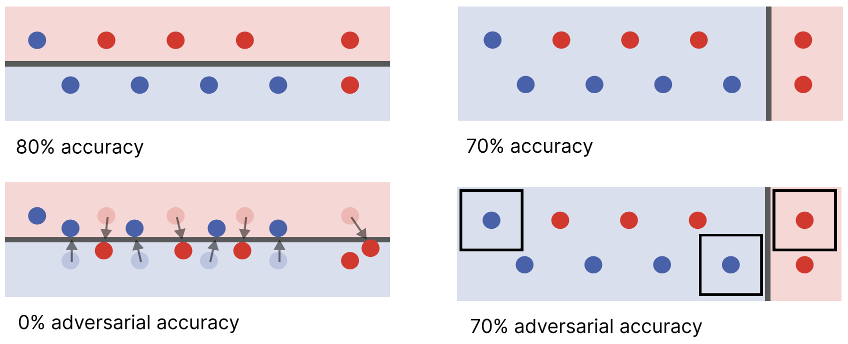


Figure 1.2: (left) the predictions of a simple decision tree that correctly classifies 80% of the red and blue samples. When the samples move by a small amount, they all get misclassified so this model is **not robust**. (right) the predictions of a **robust** tree, even when samples move within a specified box, they are still predicted correctly. Based on the example by Chen et al. [9].

Leaf Nodes Leaf nodes are special nodes with no children and indicate some prediction or action to be taken. In classification problems, these nodes hold a prediction such as 'cat' or 'dog,' while in regression problems, the nodes hold a continuous value to be predicted. There are also multi-output settings in which each leaf predicts a value for each of the multiple outputs. For example, in Chapter 10, we use multi-output regression trees that predict a probability for each available action in the problem.

1.1.2 Decision Tree Ensembles

Aside from their desirable interpretability properties, decision trees are also popular due to their success in ensembles. An ensemble is a collection of machine learning models whose predictions are combined into one final prediction to create a more powerful model. Consequently, ensemble models can be much harder to interpret, and depending on the number of models in the ensemble, they trade off interpretability for predictive performance. Depending on the use case, it can be desirable to have a single interpretable decision tree or increase performance at the cost of interpretability with tree ensembles. Random forests [4] and gradient-boosted decision tree ensembles [5–7] are particularly popular ensemble methods. These ensembles often outperform neural network-based models on tabular data prediction tasks since they can easily learn highly discontinuous patterns [8].

1.1.3 Adversarial Robustness

Machine learning algorithms have been applied to many different use cases. Unfortunately, they can be brittle in the presence of noise. For example, when applying an imperceptible amount of noise to inputs of neural networks their predictions can drastically change [10]. The same problem applies to decision trees and their ensemble, see Figure 1.2. To alleviate this problem, much research has gone into achieving adversarial robustness. In this adversarial learning setting, a model is not scored by its performance on the collected clean data, but on data with perturbations [11]. These perturbations are adversarial in the sense that we assume an adversary gets to perturb the samples in a way that they

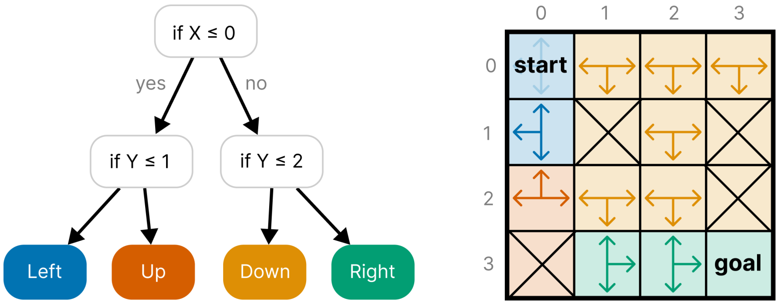


Figure 1.3: A decision tree for a **sequential decision making** problem. One has to traverse the maze from start to finish without hitting the crosses, the maze walls can be hit without penalty. The actions left, right, up, and down randomly move the agent in a direction similar to the intended action. Decision tree algorithms for supervised learning cannot solve this type of problem.

cause the model to make bad predictions. The adversary is limited by some user-specified constraints that encode their uncertainty in the collected data, usually a small ball of fixed radius around each data point.

Poisoning Robustness Another kind of robustness considers data poisoning. In such attacks, an adversary is assumed to completely remove data points or add new data to the dataset. Such attacks can happen in the real world when data is collected from the internet, for example when training large language models it is common to collect data from online forums that users can add data to. To be poisoning robust we usually assume the model should not change significantly when a small percentage of the training data is changed. Decision trees and tree ensembles suffer from both adversarial perturbations and data poisoning attacks.

1.1.4 Sequential Decision Making

Our previous mentions of machine learning have only considered supervised learning settings. In supervised learning the algorithm receives a dataset with its associated labels and needs to learn to predict the labels for unseen data. Sequential decision making problems are significantly different from this setting, instead of receiving pre-collected data, the algorithm either gets a mathematical description of an environment [12] (planning) or gets to collect its own data from the environment [13] (reinforcement learning). The goal is then to maximize some objective by making a series of predictions inside of such an environment. In this dissertation, we consider the problem of learning a policy (a function) that assigns an action to each input, an example is given in Figure 1.3. Specifically this policy is a small decision tree so it is easy for humans to interpret. Sequential decision making problems are challenging to solve due to the fact that the environments are often non-deterministic (they contain random effects) and require sequences of interdependent predictions.

1.2 Research Goal

Although size-limited decision trees have the desirable property of interpretability, we have not yet considered how these models are optimized. There are highly successful heuristics [14, 15] and optimal methods [16–20] for decision trees in supervised learning tasks such as classification and regression, but they cannot be directly applied outside this setting. At a high level, this dissertation aims to develop methods for optimizing decision trees in settings where existing methods cannot be applied or do not scale. The settings we consider are robust optimization and sequential decision making problems. What makes these settings significantly harder is that predictions in one part of the decision tree influence what predictions should be made in other parts of the tree to achieve good performance [20]. Our main goal results in 5 research questions about optimization methods for decision trees:

How to learn robust decision trees?

1. How can we efficiently optimize decision trees for robustness against adversarial examples?
2. How can we find provably optimal robust decision trees against adversarial examples?
3. How can we optimize decision trees for robustness against data poisoning attacks? (train-time versuses test-time robustness)

How to learn decision trees as policies for Sequential Decision Problems?

4. How can we find provably optimal decision tree policies for non-deterministic planning problems?
5. How can we optimize decision tree policies in reinforcement learning settings without imitating neural networks?

We have answered these questions by developing algorithms that optimize decision trees for robustness and sequential decision making leading to the contributions below.

1.2.1 Contributions

For question 1, we did this by analytically solving the split scoring functions used in robust decision tree learning heuristics, which decreased the split scoring time complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$. We also considered robustly optimizing only the leaves of a decision tree by keeping the structure of the tree (the splitting nodes) fixed. This results in a polynomial-time algorithm for binary classification problems and can be applied to arbitrary perturbation norms.

To tackle question 2, we translated the problem of optimizing robust decision trees into a Mixed-Integer Programming Formulation. This resulted in the first published method for training optimal robust trees against adversarial examples.

We answered question 3 by using differential privacy. With this technique developed to improve data privacy, we could train decision trees that provide a robustness guarantee against data poisoning. We improved the performance of differentially-private trees by adapting the algorithm based on the amount of available data.

For question 4, we translated the optimization problem of finding optimal decision tree policies for Markov Decision Processes into Mixed-Integer Linear Programming. The resulting method can find the best tree for a given size limit on a fully specified MDP.

Our algorithm for answering question 4 could not be directly used for question 5 since, in reinforcement learning, the MDP is assumed to be unknown. We developed a method based on policy gradients to optimize a decision tree policy heuristically. The key idea is to incorporate gradient information into the decision tree without explicitly differentiating it.

1.3 Outline

The rest of the dissertation contains a more comprehensive introduction to decision tree optimization in **Chapter 2** and is then followed by two main parts: **Part I** concerns the problem of optimizing decision trees for robustness, and **Part II** focuses on decision tree optimization in sequential decision making problems. Each of the parts start with an introduction chapter (Chapters 3 and 8) that provides background knowledge. The chapters within these parts are based on the papers written during the PhD, and their background sections are combined in the introduction chapters. Some published papers were omitted from the main content. The complete **list of papers** can be found in 11.4.

In **Part I**, we first look at how we can speed up heuristic algorithms for robust decision tree learning (**Chapter 4**), and then solve this problem to optimality **Chapter 5**. After that, in **Chapter 6**, we consider the problem of only robustly optimizing leaf predictions. Finally, we investigate data poisoning robustness in **Chapter 7**.

In **Part II**, we sequential decision making problems. In **Chapter 9**, we investigate the problem of finding an optimal decision tree that can be used as a policy in Markov Decision Processes (MDPs). This method requires full knowledge of the MDP, so we follow it up in **Chapter 10** with a reinforcement learning method that only uses the MDP as a simulator.

Finally, we end the dissertation with a discussion on important considerations, limitations, and recommendations for future work in **Chapter 11**.

Bibliography

[1] J. Porter, *Chatgpt continues to be one of the fastest-growing services ever,* (2023).

[2] M. T. Hicks, J. Humphries, and J. Slater, *Chatgpt is bullshit,* Ethics and Information Technology **26**, 38 (2024).

[3] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,* Nature machine intelligence **1**, 206 (2019).

[4] L. Breiman, *Random forests,* Machine Learning **45**, 5 (2001).

[5] J. H. Friedman, *Greedy function approximation: a gradient boosting machine,* Annals of statistics , 1189 (2001).

[6] J. H. Friedman, *Stochastic gradient boosting,* Computational statistics & data analysis **38**, 367 (2002).

[7] L. Mason, J. Baxter, P. Bartlett, and M. Frean, *Boosting algorithms as gradient descent,* Advances in neural information processing systems **12** (1999).

[8] L. Grinsztajn, E. Oyallon, and G. Varoquaux, *Why do tree-based models still outperform deep learning on typical tabular data?* in *Advances in Neural Information Processing Systems*, Vol. 35, edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Curran Associates, Inc., 2022) pp. 507–520.

[9] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, *Robust decision trees against adversarial examples,* arXiv preprint arXiv:1902.10660 (2019).

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. B. Estrach, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks,* in *2nd International Conference on Learning Representations, ICLR 2014* (2014).

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards deep learning models resistant to adversarial attacks,* arXiv preprint arXiv:1706.06083 (2017).

[12] M. L. Puterman, *Markov decision processes,* Wiley Series in Probability and Statistics (1994).

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction* (MIT press, 2018).

[14] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees,* (1984).

[15] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[16] S. Nijssen and E. Fromont, *Mining optimal decision trees from itemset lattices,* in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (2007) pp. 530–539.

[17] D. Bertsimas and J. Dunn, *Optimal classification trees,* Machine Learning **106**, 1039 (2017).

[18] S. Verwer and Y. Zhang, *Learning optimal classification trees using a binary linear program formulation,* in *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33 (2019) pp. 1625–1632.

[19] S. Aghaei, A. Gómez, and P. Vayanos, *Strong optimal classification trees,* arXiv preprint arXiv:2103.15965 (2021).

[20] J. van der Linden, M. de Weerdt, and E. Demirović, *Necessary and sufficient conditions for optimal decision trees using dynamic programming,* in *Advances in Neural Information Processing Systems*, Vol. 36, edited by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Curran Associates, Inc., 2023) pp. 9173–9212.

2 Introduction to Decision Tree Optimization

Decision tree learning is the problem of finding a good decision tree model for a dataset. The most established algorithms for decision tree learning have been developed for classification and regression tasks, i.e., tasks where the decision tree predicts a class label or number based on some given information. This chapter introduces the heuristic algorithms for classification and regression tree learning that are commonly used today and also presents lesser-known methods for learning optimal decision trees. Understanding decision tree optimization for classification and regression gives a good foundation for understanding the methods for robust optimization and sequential decision-making in the rest of this dissertation.

2.1 Classification and Regression

Classification and regression problems have been studied for many decades. In classification, we are given some information about an instance, and we have to predict the class that this instance belongs to. For example, given the weight and height of an animal, predict whether this animal is a cat or a dog. The pieces of information about an instance, such as weight and height in our example, are called the features, and the labels 'cat' and 'dog' are called the classes. Formally, the classification learning problem concerns finding a classifier $C : \mathbb{R}^m \to \{0, ..., (k-1)\}$ that maps the input represented as a vector of m features $x \in \mathbb{R}^m$ to one of $k > 1$ classes. We often refer to binary classification as classification problems for two classes $k = 2$ and multiclass classification as $k > 2$.

Regression is a similar problem to classification, except where classifiers predict a discrete class, regressors predict a real number. One example of a regression problem is predicting the age of an animal given their weight and height. Formally, in regression, we are concerned with finding a regressor $R : \mathbb{R}^m \to \mathbb{R}$ that accurately predicts a real number target from an input.

2.1.1 Learning as an Optimization Problem

Supervised machine learning is the problem of identifying a good model, e.g., a classifier or regressor, from a set of prerecorded instances labeled with their associated target output, which we usually refer to as the training set. A key aspect of supervised learning is determining the quality of a model because once we have a measure of quality, we can use optimization methods to maximize the measure. For example, for a classifier, we can determine the quality of the model by counting the number of instances for which the classifier assigns the correct label as determined by the training set, which is often referred to as accuracy. For regression, we could measure how much the regressor's predictions differ from the true targets, which is often referred to as the absolute error.

Generally, we measure model quality with a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ that maps a model's prediction for an instance ($\mathcal{Y}$ denotes the space of predictions) and the instance's target to a value that represents the cost of this prediction, where lower costs mean better predictions. The goal of learning is then to identify a model (e.g. a classifier or regressor) h^* from a hypothesis class (a set of possible models) $\mathcal{H}$ that minimizes the expected loss over the true distribution of X and Y, the samples and labels:

$$h^* = \operatorname*{arg\,min}_{h \in \mathcal{H}} \mathbb{E}_{x,y}[L(h(x), y)]. \tag{2.1}$$

Unfortunately, this is generally impossible as we rarely know the true joint distribution $P(x, y)$. Therefore in this dissertation, and typically in the field of machine learning, we will focus on empirical risk minimization: minimizing the expected loss on a finite sample of training data. That means that all supervised learning algorithms in this dissertation either explicitly or implicitly minimize the expected loss on the training set,

$$h^* = \operatorname*{arg\,min}_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i), \tag{2.2}$$

where n is the number of samples in the training data. The essence of developing learning algorithms is to find methods that efficiently minimize the expected loss. Since the choice of loss function L significantly influences the difficulty of the optimization problem, we will briefly introduce some important loss functions for classification and regression.

Loss Functions for Classification

For classification tasks, an interesting value to optimize is often the accuracy, i.e., the percentage of samples that are correctly classified by the model. Unfortunately, many algorithms require the loss function to be differentiable with respect to the predicted value $h(x)$, which is why these methods optimize smooth surrogate losses instead. As we will see later, most decision tree learning algorithms *can* directly optimize non-differentiable metrics such as accuracy. Below we give two important loss functions for binary classification, i.e. the true class $y \in \{0,1\}$ and predicted probability $\hat{y} = h(x) \in [0,1]$:

- The 0-1 loss $L_{0-1}(\hat{y}, y) = \mathbb{I}[\hat{y} = y]$ counts misclassifications, and minimizing it is analogous to maximizing accuracy. This loss function is not differentiable at $\hat{y} - y$ and has 0 slope elsewhere.
- The cross-entropy or log loss $L_{\log}(\hat{y}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$ treats $\hat{y}$ as a probability of predicting 0 or 1. This loss function is differentiable and monotonic w.r.t. $\hat{y}$.

Loss Functions for Regression

In regression, the targets $y \in \mathbb{R}$ and $\hat{y} \in \mathbb{R}$ are continuous values. Two common loss functions are:

- The absolute error loss $L_{\mathrm{AE}}(\hat{y}, y) = |\hat{y} - y|$ minimizes the difference between the prediction and target. This loss function is not differentiable at $\hat{y} = y$.
- The squared error loss $L_{\mathrm{SE}}(\hat{y}, y) = (\hat{y} - y)^2$ minimizes the squared difference between the prediction and target. This loss function *is* differentiable but, compared to the absolute error, has the disadvantage that it is sensitive to outliers, i.e., samples with target y far from $\hat{y}$ dominate the expected loss.

When we interpret the inputs to the squared error loss L_{SE} as probabilities of binary classes ($y \in [0,1]$ and $\hat{y} \in [0,1]$) it is analogous to maximizing the Brier score, i.e. it is the equivalent of the mean squared error for classification. As we will see later, the most popular learning algorithm for classification implicitly optimizes the Brier score.

2.2 Greedy Decision Tree Learners

In the 1980s, the decision tree learning algorithms CART [1] and ID3 [2] were published and have become hugely successful since. These algorithms are greedy, meaning that in every step of the algorithm they make the choice that results in the largest gain at that moment. Although the methods are heuristics based on a greedy algorithm, and thus cannot guarantee optimality, they perform well in practice and run very efficiently, which explains their widespread adoption. In this section, we will go through a detailed explanation of the algorithms and generalize their principles into a meta-algorithm for greedy decision tree learning.

2.2.1 Classification and Regression Trees

The decision tree learning algorithms proposed by CART [1] greedily split up the dataset one node at a time. For every node, they score all possible splits in the dataset with a criterion and pick the best one. Then, they split the dataset according to the best split and continue the splitting procedure for the left and right subtrees of the newly created node. The algorithm contains stopping criteria to decide when to stop splitting subtrees further and create a leaf node. Usually, this is done when the leaf is 'pure', i.e. it contains only samples with the same target, when the size of the tree has met a user-specified limit, and/or when the split / leaf would be too small, e.g. it would contain fewer than 10 samples. The pseudocode for classification and regression trees is given in Algorithm 1.

The CART algorithm is recursively defined and knows two cases: if the tree size has reached a limit or y fulfills some stopping properties it creates a leaf node, otherwise CART splits the dataset by creating a decision node. When creating a leaf for a classification tree CART sets the prediction value to the majority class that appears in the leaf, this maximizes the accuracy of the leaf. For regression trees minimizing mean *squared* error the leaf's prediction is the mean of targets in the leaf. While CART does not define a splitting criterion based on the mean *absolute* error, modern implementations do often support it [3], and in this case, one minimizes the mean absolute error by setting the prediction to the median of the targets.

When creating a decision node, CART searches through all possible feature and threshold combinations to find a split that best partitions the targets of the dataset. To identify all possible thresholds, CART first sorts every feature's values and selects the thresholds that are right in the middle between two consecutive values. Determining splits like this leads to a 'maximal margin' around the thresholds which is intuitively a good idea since it leaves room for uncertainty around the samples and also has theoretical motivations [4].

CART determines the quality of splits by measuring the quality of the two leaves that the split would produce by their 'impurity' values. The impurity measures how homogeneous a set of targets is. The impurity values of the leaves are computed both for the left and right subsets of samples and then averaged and weighted by the percentage of samples that are in the subset (i.e. by the probability of being in that leaf). The weighted score function for targets in the left subset y_l and targets in the right subset y_l is:

$$S(y_l, y_r) = \frac{|y_l|}{|y_l| + |y_r|} I(y_l) + \frac{|y_r|}{|y_l| + |y_r|} I(y_r), \tag{2.3}$$

where $I(y')$ is a function that measures impurity in a set of targets y'. When training classification trees CART uses the Gini impurity for I. The Gini impurity measures the probability of misclassifying a sample from the set when predicting classes according to the probability with which they appear in that set. Define the proportion of samples in y' with label k as $p_k = \frac{|\{y_i : y_i = k\}|}{|y|}$, then the Gini impurity is defined as:

$$I_{\text{Gini}}(y') = \sum_k p_k(1 - p_k) = \sum_k p_k - p_k^2 = 1 - \sum_k p_k^2. \tag{2.4}$$

The ID3 [2] algorithm uses entropy as a measure of impurity instead, but the Gini impurity and entropy functions rarely lead to different splits [5]. Often, work on decision trees will refer to information gain or impurity reduction that can be generally written as:

$$IG(y_l, y_r) = I(y_l \cup y_r) - S(y_l, y_r), \tag{2.5}$$

i.e. the improvement in 'impurity' after partitioning the data. Maximizing gain is equivalent to minimizing S. When training regression trees with CART, the impurity function is replaced by a measure of the variance within targets y'. Minimizing the variance of the targets is equivalent to minimizing the mean squared error when predicting the mean of the targets that reach each leaf. Let us now walk through an example of how to apply CART to a binary classification problem.

Example 1. *Consider a dataset with samples* $X = \{(0.0,1.0),(0.11,0.11),(0.22,0.56),(0.33,0.44),(0.44,0.22),(0.56,0.89),(0.67,0.67),(0.78,0.33),(0.89,0.78),(1.0,0.0)\}$ *and their binary labels* $y = \{0,0,0,0,0,1,1,1,1,1\}$ *visualized in Figure 2.1* ①.

We can think of the CART algorithm as starting with a tree containing a single leaf and iteratively splitting up this leaf to improve the tree. To start splitting, we find sort each feature j *and determine its maximal-margin thresholds:* $V_0 = (0.06,0.17,0.28,0.39,0.50,0.61,0.72,0.83,0.94)$ *and* $V_1 = (0.06,0.17,0.28,0.39,0.50,0.61,0.72,0.83,0.94)$ *as visualized in step* ②. *we compute for each feature* j *and for each possible threshold value* $v \in V_j$ *the score function* S *(using Equation 2.3) with the Gini impurity and store the* j^*, v^* *that minimize* S. *We compute (and display in* ②*) the* S *values for all possible splits, and we find that the optimal split is given by* $j^* = 0, v^* = 0.28$ *with a value for* S *of* 0.29. *Note that both* $(j = 0, v = 0.28)$ *and* $(j = 1, v = 0.72)$ *minimize* S *so we chose one of these solutions arbitrarily.*

In step ③, *we recursively continue the CART algorithm on the region left of* 0.28 *for feature* 0 *(x-axis) and the region right of it. On the left side, we find that the set of targets is completely pure since it only contains (red) samples with* $y_i = 0$. *Therefore we create a leaf and do not recurse on that side. On the right side, we do not satisfy the stopping criteria, so we continue splitting. We identify the possible split thresholds* $V_0 = (0.39,0.50,0.61,0.72,0.83,0.94)$ *and* $V_1 = (0.11,0.28,0.39,0.56,0.72,0.83)$, *and find that the optimal split is given by* $j^* = 1, v^* = 0.72$ *with a value for* S *of* 0.0.

In step ④, *we recurse on the regions left and right of* 0.72 *for feature* 1 *(y-axis). In both cases, we find that the sets of samples are completely pure and have, therefore, satisfied one of the stopping criteria. We create a leaf for both sides and end up with a final decision tree of 2 nodes and 3 leaves. The resulting decision tree accurately predicts the training dataset.*

2.2.2 Different Branch Node Predicates

Categorical Features

Unfortunately, many implementations of machine learning algorithms only support numerical feature values, so it can be interesting to consider changing categorical features into numerical ones. One naive approach would be to encode a categorical feature's k categories into numbers $0..k$. For example, for a categorical feature 'language,' we could encode its values as 'Dutch' = 0, 'French' = 1, and 'English' = 2. While this enables training on categorical features with algorithms that only support numerical features, a disadvantage is that it introduces an arbitrary ordering of the categories. In our example, Dutch and English are now less similar than Dutch is to French, which is an undesirable effect as there is usually no clear notion of distance between discrete categories.

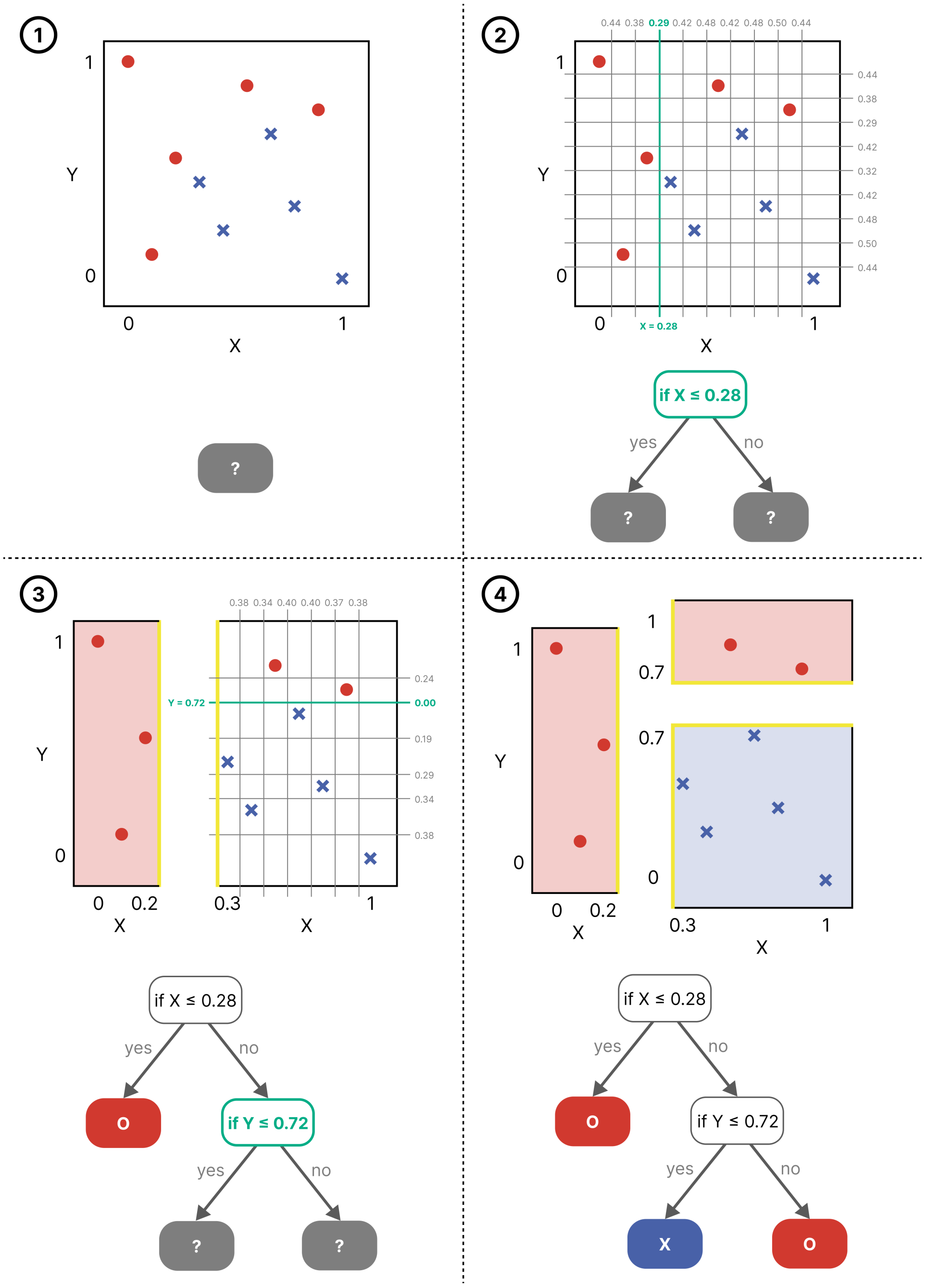


Figure 2.1: Visualization of Algorithm 1 running on a simple binary classification task in two dimensions. By greedily adding nodes to the tree according to the best Gini impurity and creating a leaf when the node is pure (only contains samples with the same label), we end up with a decision tree that fits the data well.

Algorithm 1 Classification and regression trees

```
1: procedure FitDecisionTree($X, y$)
2:     if stopping criterion (e.g. maximum depth/number of nodes, pure leaf) then
3:         return Leaf($y$)
4:     end if
5:     for $j = 0...m$ do
6:         let $Z$ be the sorted set of unique feature values $X_{ij}$ for $i = 0...(n-1)$
7:         compute all threshold values $V_j = \{... , \frac{1}{2}(Z^{(i)} + Z^{(i-1)}) , ...\}$ where $(Z^{(i)} > Z^{(i-1)})$
8:     end for
9:     $j^*, v^* = \arg\min_{j,v \in V_j} S(\{y_i : X_{ij} \leq v\}, \{y_i : X_{ij} > v\})$        ▷ see Equation 2.3
10:    $T_l =$ FitDecisionTree($\{X_i : X_{ij^*} \leq v^*\}, \{y_i : X_{ij^*} \leq v^*\}$)
11:    $T_r =$ FitDecisionTree($\{X_i : X_{ij^*} > v^*\}, \{y_i : X_{ij^*} > v^*\}$)
12:    return Node($j^*, v^*, T_l, T_r$)
13: end procedure
```

When considering categorical features containing few categories (e.g., up to 10), it can be a good idea to encode the feature into multiple separate numerical features that indicate whether or not the category belongs to the instance. Such an encoding is known as one-hot encoding. This way, the feature 'language' with categorical values 'Dutch,' 'French,' and 'English' can be encoded into three numerical features 'is_Dutch,' 'is_French,' and 'is_English.' One-hot encoding allows us to use algorithms for numerical features and creates interpretable decision nodes that send instances with one category to the left subtree and all other categories to the right subtree. One disadvantage of this method is that if single categories carry little information on the target, we would need a large decision tree with many decision nodes to get good predictive results.

We will consider one more way to deal with categorical features, which is by 'natively' supporting them with a predicate for categorical values. The predicate partitions the categories into a set for a node's left and right subtree. Returning to our example, we can interpret the predicate to say: 'language ∈ {Dutch, English},' which would send the instance to the left subtree if true and to the right subtree if not true. Although dealing with categorical features this way requires a specialized algorithm, it results in trees that require fewer nodes than trees trained with one-hot encoding. As we will see later, some algorithms can efficiently support these kinds of features.

Features with Missing Data

Aside from dealing with different types of features, there is sometimes also the problem with datasets that contain missing values. Usually, these are datasets where, for some instances, a specific feature value is unknown. While there is a plethora of work on dealing with this issue in machine learning, decision trees can natively support missing values by treating them as a separate category, which is usually as effective and more efficient than, for example, imputing missing values [6]. To deal with missing values as a category we simply keep track in a decision node of whether missing values for the chosen feature move to the left or right subtree.

2.2.3 Oblique Decision Trees

In typical decision trees, the decision nodes select one feature to split on, thereby cutting up the sample space with axis-parallel splits (as seen in Figure 1.1). While this aids interpretability by making the decision tree behavior easier to understand, it also limits the expressivity of the model. Therefore some lines of work consider trading interpretability for expressivity using oblique decision trees, decision trees in which the decision nodes (and sometimes also the leaves) can use linear functions. Instead of axis-parallel cuts in the sample space, this results in arbitrary hyperplanes.

Oblique decision trees can be heuristically learned by greedily splitting up the space using linear classifiers, which is a modification of the CART algorithm [1]. Such algorithms can often get stuck in local minima, so other algorithms were proposed with randomization to overcome these local optima [7, 8]. Some Mixed-Integer Linear Programming methods (see Section 2.4.3) such as Bertsimas and Dunn [9] can produce optimal oblique decision trees by relaxing integer constraints.

There are also versions of oblique decision trees that not only allow nodes to make oblique splits but also create more powerful leaf nodes. In these kinds of trees, each leaf node holds a separate linear model that is used to arrive at the final prediction. Decision trees with oblique splits and linear models in the leaves can represent models that are equivalent to ReLU neural networks [10]. However, even though machine learning methods hold the same representation power, it does not mean that they are equivalent, as they might use different optimization methods, which results in different solutions for the same training data. Some other methods use decision trees with splits on single features but with linear models in the leaves[1].

2.3 Decision Tree Ensembles

Although singular size-limited decision trees are interpretable and can predict non-linear relationships, there are situations when an application requires a more complex model to achieve an accurate predictor. In these situations, it is common to give up interpretability for performance by combining many decision trees into an ensemble. Although neural network-based techniques dominate the field for unstructured tasks with, for example, audio and video data, for tabular datasets, tree ensembles still often outperform neural networks [11]. The field of decision tree ensembles is too broad to cover in full detail in this introduction, thus we will focus our attention on the two most popular types of tree ensembles: random forests and gradient boosting.

2.3.1 Random Forests

Random forests [12] were introduced by Breiman. The random forest algorithm trains large decision trees that perfectly predict the training data and uses two methods to ensure the diversity of the ensemble, which prevents overfitting. See Algorithm 2 for the pseudocode.

The first method to promote ensemble diversity is called bootstrap aggregation [13] (often referred to as bagging), which trains each tree on a separate bootstrap sample. Such a bootstrap sample is generated by sampling *with* replacement from the training dataset

[1]`https://github.com/cerlymarco/linear-tree`

until a new training set of the same size is gathered. Effectively, such a bootstrap dataset contains roughly two-thirds of the samples from the original dataset and contains multiple copies of some samples.

The second method that random forests use to create diversity is to limit the possible features during each iteration of the greedy splitting procedure. Every time a tree creates a new node, it randomly samples different features until it has the square root of the number of total features. The best split is then chosen from this limited set of features. Since each node can only split on a feature from the random subset of features, the probability that two trees in the ensemble use the same features is very low.

Random forests have had widespread success on tabular prediction tasks as they usually perform very well with little tuning. Modern greedy decision tree learners have extremely efficient implementations, and since all trees of a random forest can be trained in parallel, the algorithm is orders of magnitude faster to train than a neural network. One disadvantage of random forests is that they consist of tens or hundreds of large decision trees, which does not allow for interpretation and makes automated verification quite challenging. Particularly for datasets with millions of samples and many unique feature values, each tree can contain thousands of nodes when hyperparameters are not tuned. Random forests are often outperformed by well-tuned gradient boosting ensembles on large datasets. Still, random forests are commonly used in situations when the number of samples per feature is low. For example, random forests are widespread in biological fields where data is gathered from expensive clinical trials or rarely observed phenomena.

Hyperparameters

By default, random forest implementations such as the one in Scikit-learn [3] train 100 trees that are unrestricted in size, which generally works well and requires little tuning. However, on large and complex datasets, this can lead to impractically large models. One simple improvement is to set the minimum number of samples per leaf to at least 0.1% of the dataset size. This has minimal impact on the final performance and reduces the number of leaves. Also, as we will see in Part I it reduces the fragility of the model. If the resulting random forest is still too large, then the trees can be limited in the number of nodes or depth, but this often comes at a cost in predictive performance.

Why Random Forests Work

Traditionally, the success of random forests has been explained in terms of the bias-variance trade-off [14]. Specifically, the choice to train decision trees that perfectly classify the training set introduces low bias, and by averaging multiple trees whose covariance is low (due to the diversity), the variance is reduced. Very recently, the adaptive smoothing effect of random forests has been studied [15] as a richer explanation of the effectiveness of random forests and gradient boosting ensembles.

Algorithm 2 Random Forest

```
1: for $m = 1...M$ do
2:     sample a bootstrap dataset $X_m$, $Y_m$ of size $|X|$ *with* replacement
3:     *learn* tree $T_m$ on $X_m, Y_m$
4: end for
5: return $f(x) = \frac{1}{M}\sum_{m=1}^{M} T_m(x)$
```

Algorithm 3 Gradient Boosting

```
1: create leaf $f_0 = \arg\min_\theta \sum_i^N L(y_i, \theta)$
2: for $m = 1...M$ do
3:     residual $r_{mi} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$
4:     *learn* regression tree $T_m$ on $X$, $\mathbf{r_m}$
5:     update $f_m(x) = f_{m-1}(x) + \alpha T_m(x)$
6: end for
7: return $f(x) = T_0(x) + \alpha \sum_{m=1}^{M} T_i(x)$
```

2.3.2 Gradient Boosted Decision Trees

Gradient-boosted decision trees are currently among the most popular machine learning methods and typically outperform neural networks on tasks involving tabular data [11]. Gradient boosting [16–18] works by iteratively adding weak learners to the ensemble that compensate for the errors made by the preceding learners. Modern implementations such as XGBoost [19], LightGBM [20], Catboost [21], and HistGradientBoosting (Scikit-learn) [3] are highly optimized and have achieved top scores in machine learning competitions[2]. Many of these frameworks offer native support for mixed numerical, categorical, and missing data which is important for many real-world applications.

Learning Algorithm

Gradient boosting algorithms start with an initial predictor and improve its predictions by iteratively adding decision trees that compensate for the residual error of the preceding models. The pseudocode for the algorithm is given in Algorithm 3. An advantage of gradient boosting over other algorithms for classification and regression is that its generalized formulation allows for optimizing arbitrary differentiable loss functions. This way, we can train a regressor by minimizing the mean squared error, train a classifier by minimizing the cross-entropy loss, and train rankers using a specialized loss function, for example. We will assume that the user provides us with a differentiable loss function $L(x, f(x))$.

To start the algorithm, it is common to optimize a decision tree that consists of a single leaf with value θ by optimizing $\theta = \arg\min_\theta \sum_i L(y_i, \theta)$. For some specific loss functions, such as the mean squared error, we can directly compute $\theta = \bar{y}$ as the mean of the targets y. After initializing the ensemble, in every iteration m, we compute the pseudo-residual $r_m i = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$ for every sample x_i as the negative gradient of the loss with respect to the predictions made by the current ensemble. We then learn a regression tree to predict these residuals and add it to the ensemble weighted by a factor α. The non-negative learning rate α is usually chosen to be a small constant such as 0.1 to avoid making excessively large changes to the ensemble in an iteration. After M iterations or a stopping criterion, such as a lack of improvement in the validation loss, the boosting process is terminated. Finally, predictions are made by summing the α-weighted tree predictions to the initial leaf prediction.

[2]Kaggle (`https://www.kaggle.com/competitions`) is a well-known organization that hosts a variety of machine learning competitions, winning submissions for tabular competitions often use gradient boosting techniques.

Histogram-based Tree Learners

Since gradient boosting ensembles are used to train sometimes hundreds or thousands of decision trees on datasets with millions of samples there has been a need to make the decision tree learners more efficient. Frameworks such as XGBoost have pioneered methods that use histograms inside of the decision tree learning algorithm to make the splitting procedure extremely fast. Typical decision tree learners consider a split at every unique feature value, but when scaling up to millions of samples, with potentially millions of possible split locations, this becomes inefficient and probably unnecessary. Histogram-based algorithms consider limiting the number of possible splits per feature to a fixed value, such as 255, and then pre-process the dataset by assigning each feature value into one of the resulting bins. Specifically, feature value histograms improve runtime efficiency since they do not require sorting each feature of the dataset and can be partitioned after splitting with fewer operations.

While primarily motivated by runtime efficiency for greedy learners, histogram-based tree learners can also be useful in other algorithms. For example, in our work on differential privacy, we will see that limiting the number of possible splits with fixed-size histograms leads to a better privacy-performance trade-off. Also, limiting the number of splits for optimal decision tree methods removes the optimality guarantee but, in practice, gives better trees than the greedy methods and is much faster than proving optimality on all possible splits.

Algorithmic Improvements

The pseudocode for gradient boosting in Algorithm 3 gives a high-level procedure but does not contain many of the algorithmic improvements that exist in many modern implementations. While there are too many to list, we will discuss a few notable ones:

- XGBoost uses an approximation of the hessian of the loss function to improve the iterative updates. The second-order gradient information can be used to reduce relative step sizes when the curvature of the loss is high to avoid overly large steps. Methods that do not use second-order derivatives often perform an extra line search step to individually weigh the leaves in every update.

- Many frameworks support categorical values natively, by partitioning categories into a left and right subset trees can make more informative splits than by simply splitting on categories one at a time with one-hot encoding.

- By considering missing numbers as a separate value, many frameworks allow the learner to choose whether missing values belong to the left or right subtree at each node. This way of handling missing values is much more efficient than imputing missing values and often outperforms it in predictive performance when the fact that a value is missing carries information, i.e. when the missingness of values is not truly random.

- Multiple frameworks offer GPU support with specialized algorithms that allow for greater parallelization. This can speed up the algorithm significantly on very large datasets.

- All frameworks offer a large number of hyperparameters that can be tuned to regularize the models or offer a trade-off between runtime and predictive performance.
- Some frameworks offer options to set constraints that enforce properties such as monotonicity. For example, one can encode knowledge that an increase in a feature's value can only result in a decrease/increase in the target value. Incorporating domain knowledge into the training procedure reduces the number of data points that a model needs to achieve good performance by restricting the number of possible models that fit the data.

Hyperparameters

Every framework contains numerous specific hyperparameters, but the most common and significant ones for tuning are worth mentioning. The predictions of gradient boosting ensembles slowly get more and more different from constant predictions as more trees are added. Therefore to control the effective complexity of the ensemble, the combination of the learning rate α and the number of trees in the ensemble is vital to tune. Usually, α is tuned, and the number of trees is determined by stopping the iterative learning procedure when performance on a held-out validation set does not improve.

Another important hyperparameter is the size of each tree that is added, which can be controlled by limiting the number of nodes or limiting the depth (which implicitly also limits the number of nodes) of the tree. By limiting the size of each decision tree, the ensemble is more regularized. While the exact effect of a limit on the number of nodes is hard to interpret, limiting the depth intuitively limits the number of possible interacting features to the depth of the tree.

Lastly, the specific loss function that the model optimizes is an important modeling choice as well. There is an abundance of possible loss functions varying in properties such as sensitivity to outliers, convergence rate, and computational efficiency. Depending on the use case of the resulting tree ensemble different loss functions are appropriate.

2.3.3 Other Tree Ensembles

While random forests and gradient boosting ensembles are currently the most popular algorithms for training decision trees, there are some other methods worth mentioning. AdaBoost [22] is one of the earliest boosting methods and a foundation of algorithms for gradient boosting. It is typically used to train a series of decision stumps (decision trees of depth 1) that improve on the mistakes of preceding stumps by reweighing the samples. Extremely randomized trees [23] is an ensemble of decision trees whose splits are chosen completely at random. These trees can be created very efficiently, but using random splits comes at the cost of predictive performance.

Tree ensembles have also been applied to unsupervised learning tasks; we reference some notable works. Isolation forests [24] are models for anomaly or outlier detection and also use random splits to create the decision trees in the ensemble. Isolation forests compute an outlier score for a point by estimating how many random splits are necessary to separate that point from the training dataset completely. Extended isolation forests [25] use oblique decision trees instead of typical axis-aligned decision trees, which can improve performance on some tasks. Half-space trees [26] use techniques like isolation forests to create anomaly detectors for streaming data.

2.4 Optimal Decision Tree Learners

While greedy decision tree learners are successful, they do not find the most accurate models. They can even get stuck on specific datasets, leading to the same performance as random guessing while perfectly accurate models exist. Unlike heuristic decision tree learners such as CART (Section 2.2), optimal decision tree learners provably find the tree that achieves the best possible score on the training data. In this section, we will introduce Mixed-Integer Linear Programming-based decision tree learners. We currently limit ourselves to classification trees, but many methods could be extended to a regression setting with modest changes. We end with a short reflection on when which paradigm is favorable.

2.4.1 Mixed-Integer Linear Programming

Linear programs (LPs) consider mathematical optimization problems that optimize a linear function of continuous variables under a set of linear constraints. These problems are generally formulated in their standard form

$$\begin{aligned} \max \quad & \mathbf{c}^\top \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned}$$

where $\mathbf{c}$ is a fixed vector of weights, matrix A and vector $\mathbf{b}$ represent a set of linear constraints and the vector $\mathbf{x} \in \mathbb{Q}^n$ holds the decision variables. These optimization problems have been studied for many decades and, without going into extensive detail, can be solved up to huge scales with simplex [27] or interior point methods [28]. Although popular methods such as the simplex algorithm take worst-case exponential time, there are interior point methods for linear programs that run in polynomial time.

Mixed-integer linear Programming (MILP) is an extension of linear programming that restricts the set of possible values for some variables to integers. Whereas LPs represent convex problems, MILPs are non-convex optimization problems and are expected to *not* be polynomial-time solvable as they are NP-hard. However, with decades of algorithmic improvements for MILP solving, we have modern solvers that can scale up to large instances. A typical notation for MILPs is:

$$\begin{aligned} \max \quad & \mathbf{c}^\top \mathbf{x} + \mathbf{h}^\top \mathbf{y} \\ \text{s.t.} \quad & A\mathbf{x} + G\mathbf{y} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \\ & \mathbf{y} \geq 0 \\ & \mathbf{y} \in \mathbb{Z}^n. \end{aligned}$$

Which is simply the LP standard form with extra variables $\mathbf{y}$ that are forced to take integer values. Most formulations for optimal decision trees do not permit the values $\mathbf{y}$ to take any integer value but further limit $\mathbf{y}$ to take a binary value, i.e., $\mathbf{y} \in \{0,1\}^n$. A MILP with only integer variables is called an Integer Linear Program (ILP), and a problem with only binary variables is called a 0-1 linear program. Note that although the decision variables $\mathbf{y}$ must take integer values, the coefficients $\mathbf{c}$, $\mathbf{h}$, A, G and $\mathbf{b}$ can still take rational values.

MILP solving

A key part of MILP solving involves the LP relaxation, i.e. the LP we get when we allow variables $\mathbf{y}$ to take rational values by relaxing the integer constraint. An important property of the LP relaxation is that it never disallows a solution to the original MILP, which means that the objective value of its optimal solution is never worse than that of the MILP. Therefore solving the LP relaxation naturally gives a bound on the objective value of the original MILP. MILPs are usually solved with a combination of branch-and-bound and cutting plane techniques that both make heavy use of the LP relaxation.

At a high level, branch-and-bound [29] splits up the problem into two branches by making an assumption on some variable (for example, $x_i \leq 5$ and $x_i \geq 6$) and solves the relaxed problems to get a bound on the objective value of both branches. If one of the branches has a worse relaxed objective than the current best solution, then it can be safely ignored. Otherwise, the branch-and-bound procedure can be continued until the LP relaxation gives us a solution that satisfies the integer constraints.

Another important technique in MILP solving is cutting plane generation. The idea is to solve the LP relaxation and if the resulting solution does not satisfy the integer constraints, introduce a new constraint (a cutting plane) that is guaranteed to be valid for all integer solutions but that invalidates the current non-integer solution. While it is possible to solve MILPs with cutting planes alone [30], the technique is more effectively used in combination with branch-and-bound [31] to produce 'branch-and-cut' techniques.

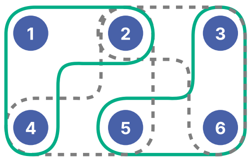
MILP formulations in practice

The runtime required to solve MILP instances increases exponentially in terms of the size of the formulation. However, aside from formulation size, there are other practical considerations that can have a significant effect on solving times. Below are some tips for formulation design that usually improve solving time:

- Introduce extra constraints to remove symmetries. When multiple different variable assignments can represent the same object inside the formulations, the solver might have to branch multiple times to prove the optimality of a solution. In the cases where it is possible to add a few extra constraints to remove such symmetries, it is usually beneficial to do so. For example, in decision trees, it is not useful to create the same split in multiple descendant nodes, and this behavior can be disallowed by introducing extra constraints.

- Reduce the number of integer variables. Since solving the LP relaxation does not guarantee that solutions will hold integer values for integer variables, MILPs are solved by branching on integer variables. When reducing the number of integer variables, the size of the complete branch-and-bound search tree will also be smaller and will likely also reduce runtime in practice.

- Use constraints that are 'tighter' when relaxed. Tightness in the LP relaxation refers to how close LP solutions will be to the integer version of the constraints. For example, if a constraint relies heavily on variables to be integers to be valid, then relaxing it will likely result in a useless constraint. On the other hand, constraints that are still useful when relaxed will lead to tighter bounds and, therefore, improved solve times (since better bounds allow more search tree nodes to be pruned).

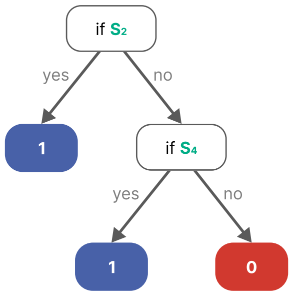


$S_1 = \{2, 4, 5\}$
$S_2 = \{3, 5, 6\}$
$S_3 = \{2, 3, 6\}$
$S_4 = \{1, 2, 4\}$

(a) Exact cover instance

X				Y
S_1	S_2	S_3	S_4	
0	0	0	1	1
1	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	1
0	1	1	0	1
0	0	0	0	0

(b) Optimal decision tree instance



(c) Optimal decision tree solution

Figure 2.2: An example instance of the Exact Cover by 3-sets problem (with the solution highlighted) and its corresponding decision tree optimization problem: find a tree with two branch nodes that correctly classifies all data. By reducing the NP-complete exact cover problem to an optimal decision tree problem, the optimal decision tree problem is proven to be NP-complete.

- Prevent numerical problems due to float imprecision. While there exist exact solvers, practical solvers use imprecise floating point numbers to represent values since these are much more efficient. However, when a MILP formulation contains both extremely large and extremely small numbers, the solver has to be executed with reduced tolerances to prevent numerical errors, which slows down execution significantly. It can, therefore, be more beneficial to formulate problems with more integer variables if this reduces numerical instability.

2.4.2 NP-completeness

We give a short proof of the NP-completeness of decision tree optimization inspired by the proof of Hyafil and Rivest [32]. The proof is based on a reduction from the Exact Cover by 3-Sets (X3C) problem to the decision problem 'does there exist a decision tree of at most k nodes that correctly classifies all data points?'.

Theorem 1. *Constructing a binary decision tree of at most k nodes that correctly classifies all training data $X \in \{0,1\}^{n\times m}$ with labels $Y \in \{0,1\}^n$ is an NP-complete problem.*

Proof. We will first show that the problem is in NP and then prove that it is NP-hard using a reduction from the Exact Cover by 3-Sets. (X3C) problem.

Verifying a solution to the problem can be done in polynomial time by computing the decision tree's predictions for each of the n rows of X and checking if these are correct with respect to Y. This takes time linear in terms of n and the size of the tree. Therefore the problem is in NP.

The exact cover problem is one of Karp's 21 NP-complete problems [33], and its variant Exact Cover by 3-Sets (X3C) with sets of exactly size 3 is also NP-complete [34]. Given a set of triples $\mathcal{S} = \{S_1, S_2, ..., S_s\}$ that each have elements from a set $Z = \{z_1, z_2, ..., z_z\}$ with z a multiple of 3, the X3C problem is to find a subset of $\mathcal{S}^* \subseteq \mathcal{S}$ that satisfies:

- The set $\mathcal{S}^*$ contains one of each element of Z, i.e. $\bigcup_{S \in \mathcal{S}^*} S = Z$.

- The sets in $\mathcal{S}^*$ do not overlap, i.e. $S_i \cap S_j = \emptyset, \quad \forall (S_i, S_j) \in \mathcal{S}^* \times \mathcal{S}^*, i \neq j$.

In the reduction from X3C to decision tree optimization, we create a binary dataset (X, Y) that encodes an exact cover instance $(Z, \mathcal{S})$ and set $k = \frac{z}{3}$ (which is an integer since z is chosen as a multiple of 3). We will create the data matrix X such that the rows (samples) correspond to elements of the set Z and columns (features) correspond to the sets $\mathcal{S}$. The binary entry X_{ij} encodes whether or not element z_i is included in set S_j. Let the label Y_i be 1 for all $i = 1, 2, ..., z$. We add one more entry x_{z+1} to X with all zero values and label $Y_{z+1} = 0$. An example of the encoding from an instance of X3C to optimal decision trees is given in Figure 2.2. This conversion can be done in polynomial time by enumerating the $(z+1) \times s$ entries of X and the $z+1$ entries of Y. To prove that the reduction is correct, we show that a yes-instance of the optimal decision tree problem implies a yes-instance of X3C and vice-versa.

Given a yes-instance for the optimal decision tree problem (X, Y, k), the solution will be a binary decision tree with at most k nodes. Each node splits off at most 3 data points at a time since each feature only contains 3 ones by construction. A solution will also split off at least 3 data points at a time because, in order to separate $z = 3k$ samples with class 1 from the sample with class 0 with nodes that split off at most 3 samples at a time, the nodes must split off at least $\frac{z}{k} = 3$ samples. Therefore the k sets indicated by the features used in the k branch nodes are an exact cover.

Given a yes-instance for the exact cover problem $(Z, \mathcal{S})$, the solution will be a subset $S \subseteq \mathcal{S})$ of cardinality $k = \frac{z}{3}$. We can construct a decision tree with k nodes (all connected by the false cases), that each use one of the features corresponding to the sets selected by S. By definition, the data rows of X corresponding to these features do not overlap, and all have label $Y_i = 1$, which means that the constructed decision tree can correctly classify all of the samples in X (with $k+1$ leaf nodes).

Since the optimal decision tree problem is in NP and there is a polynomial-time reduction from the NP-complete problem X3C to it, the problem is NP-complete. $\square$

2.4.3 Optimal Trees using Integer Programming

The papers in this section provide mathematical formulations of the decision tree optimization problem in a way that the formulations can be input into powerful MILP solvers. Recall that MILP formulations consist of a linear objective term with a set of linear inequality constraints. For more background on (Integer) Linear Programming, see Section 2.4.1. The notation used in the MILP formulations of this section is summarized in Table 2.1. In the different formulations, we highlight variables used to determine node features in blue, node threshold values in red, and leaf predictions in yellow.

Optimal Classification Trees (Bertsimas and Dunn)

In 2017, Bertsimas and Dunn published one of the most popular papers on optimal decision tree learning using Integer Linear Programming. For their formulation, from here onwards, referred to as OCT (Optimal Classification Trees), the user provides training data with p normalized feature values $\mathbf{x_i} \in [0, 1]^p$ and labels y_i for each instance $i = 1...n$. It is important that all feature values are normalized into the range $[0, 1]$ since this is later required for the validity of the constraints. OCT then finds a tree that minimizes the error

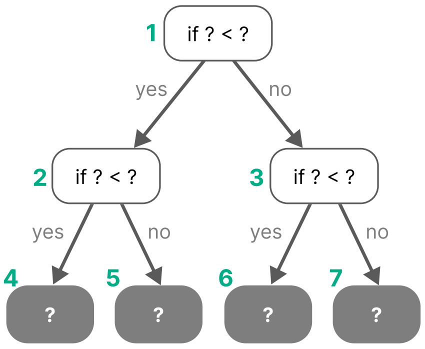


Figure 2.3: In MILP formulations such as OCT [9] the tree structure is fixed and the solver decides the features and thresholds used in the branch nodes, and the predictions in the leaf nodes. A numbering starting at 1 in the root node and going top-down, left-to-right can be conveniently used to define parent-child relations.

plus a weighted term that penalizes tree size. OCT proposes three hyperparameters for regularization:

- D: the maximum depth of the decision tree.
- N_{min}: the minimum number of samples required for creating a leaf node.
- α: the weight of the tree size regularization term in the objective.

The way OCT formulates classification tree optimization as an integer program is by fixing the structure of a full binary tree of given depth d and then assigning values to the split features, split thresholds, and leaf predictions inside of each node. That way the tree can be thought of as a template into which the decision node and prediction leaf contents will be assigned by the solver. The nodes and leaves are numbered in increasing order starting at 1 from top to bottom and left to right as in Figure 2.3. This numbering is useful since one can identify parents and children of some node t by simple arithmetic.

Using the numbering system, the tree is partitioned into two sets: the set of decision nodes $\mathcal{T}_B = \{1, ..., 2^D - 1\}$, where node 1 is the root node, and the set of leaf nodes $\mathcal{T}_L = \{2^D, ..., 2^{D+1} - 1\}$. Moreover, the notation $p(t) = \lfloor \frac{t}{2} \rfloor$ is used to indicate node t's parent and $A(t)$ the set of all ancestors of t. The set $A(t)$ is further partitioned into $A_L(t)$ and $A_R(t)$ where $A_L(t)$ contains the ancestors of t for which t is in the left subtree and $A_R(t)$ the ancestors for which t is in their right subtree.

Constraints In a typical classification tree, each node splits on a feature and an associated threshold value. In the OCT formulation, the binary variables a_{jm} indicate whether or not node m splits on feature j and continuous variables b_m hold the threshold value associated with node m. Since OCT allows decision nodes to be 'turned off,' the solver is allowed to set either all $a_{jm} = 0$ and $b_m = 0$ for node m or set one of the a variables to 1 and b_m to a non-zero value. The variables d_m indicate whether a node is 'on' or 'off'. This behavior is encoded using the constraints:

$$\sum_{j=1}^{p} a_{jm} = d_m, \qquad \forall t \in \mathcal{T}_B, \tag{2.6}$$

$$0 \leq b_m \leq d_m, \qquad \forall m \in \mathcal{T}_B, \tag{2.7}$$

$$a_{jm} \in \{0,1\}, \qquad j = 1, \ldots, p, \forall m \in \mathcal{T}_B, \tag{2.8}$$

$$d_m \in \{0,1\}, \qquad \forall m \in \mathcal{T}_B. \tag{2.9}$$

When a parent node is ‘turned off’, its children should also be turned off:

$$d_m \leq d_{p(m)}, \quad \forall t \in \mathcal{T}_B \setminus \{1\}. \tag{2.10}$$

The variables z_{it} are introduced to indicate whether data sample i reaches leaf t. The extra variables u_t then indicate whether any data sample reaches leaf t. The u_t variables are used to model the requirement that every leaf node either contains no samples or at least $N_{\min}$ samples:

$$z_{it} \leq u_t, \quad i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \tag{2.11}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{\min} u_t, \quad \forall t \in \mathcal{T}_L, \tag{2.12}$$

$$z_{it} \in \{0,1\}, \quad i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \tag{2.13}$$

$$u_t \in \{0,1\}, \quad \forall t \in \mathcal{T}_L. \tag{2.14}$$

Now we will start to model the assignment of samples to leaves. The idea behind this part of OCT is to force each sample to be assigned to exactly one leaf:

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad i = 1, \ldots, n, \tag{2.15}$$

and to prevent sample i from being assigned to leaf t when the splits in the ancestor nodes of t do not enable the sample going to t:

$$\mathbf{a}_m^\top \mathbf{x}_i < b_m + M_1(1 - z_{it}), \quad i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \forall m \in A_L(t), \tag{2.16}$$

$$\mathbf{a}_m^\top \mathbf{x}_i \geq b_m - M_2(1 - z_{it}), \quad i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \forall m \in A_R(t). \tag{2.17}$$

In effect, OCT forces $z_{it} = 0$ for each unreachable leaf t, which forces sample i to be assigned to the only reachable leaf t' ($z_{it'} = 1$) because each sample must be assigned to one leaf (Constraint 2.15).

The pair of constraints 2.16 and 2.17 are important to further discuss as these are examples of big-M constraints and generally known to have relatively loose relaxations [35]. Moreover, Constraint 2.16 requires a strict inequality, which is something that cannot be directly modeled in MILP. Therefore one first rewrites the constraint to:

$$\mathbf{a}_m^\top (\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \quad i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \forall m \in A_L(t), \tag{2.18}$$

where ϵ is chosen in such a way that it is larger than zero but smaller than any difference between two adjacent feature values (to make sure the constraint does not disallow any useful splits). Let $x_j^{(i)}$ be the ith largest value of feature j, then ϵ is defined as:

$$\epsilon_j = \min\left\{x_j^{(i+1)} - x_j^{(i)} \mid x_j^{(i+1)} \neq x_j^{(i)},\ i = 1,...,n-1\right\}. \tag{2.19}$$

Now, the big-M values have to be selected large enough that they satisfy the constraint for any setting of a_{jm} and b_m, but as small as possible to maintain the tightness of the LP relaxation. Since $x_{ij} \in [0,1]$, $a_{jm} \in [0,1]$ and $b_m \in [0,1]$, one can set $M_1 = 1 + \epsilon_{\max}$ and $M_2 = 1$ (their smallest valid values).

OCT minimizes the sum of classification errors. To count misclassifications the constant Y_{ik} is introduced which is defined as:

$$Y_{ik} = \begin{cases} +1 & \text{if } y_i = k \\ -1 & \text{otherwise} \end{cases}, \quad k = 1,...,K,\ i = 1,...,n. \tag{2.20}$$

Intuitively, it is +1 when the correct class k (out of K total classes) is predicted, and -1 otherwise.

Limitations Since the OCT formulation relies on big-M constraints the relaxations are not tight which results in long runtimes when proving optimality. Furthermore, the strict inequality of Equation 2.16 leads to the introduction of a small constant ϵ_j which means the tolerances of the solver have to be set appropriately to prevent floating point errors. With smaller tolerances the solver is further slowed down. Later MILP formulations for optimal decision trees have worked on speeding up optimization. However, OCT still provides a useful foundation for modeling.

Combined Formulation The complete MILP formulation for OCT[3] is:

[3]There are some minor differences between the OCT formulation as presented in this dissertation and the formulation in [9]. Some of these are corrections, and others are made to preserve the consistency with the rest of this dissertation.

leaf t predicts class k

$$\begin{aligned}
\min \quad & \frac{1}{L_{\text{base}}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \sum_{m \in \mathcal{T}_B} d_t & & (2.21)\\
\text{s.t.} \quad & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), & k = 1, \ldots, K, \forall t \in \mathcal{T}_L, \quad & (2.22)\\
& L_t \leq N_t - N_{kt} + n\, c_{kt}, & k = 1, \ldots, K, \forall t \in \mathcal{T}_L, \quad & (2.23)\\
& L_t \geq 0, & \forall t \in \mathcal{T}_L, \quad & (2.24)\\
& N_{kt} = \frac{1}{2} \sum_{i=1}^{n} (1 + Y_{ik}) z_{it}, & k = 1, \ldots, K, \forall t \in \mathcal{T}_L, \quad & (2.25)\\
& N_t = \sum_{i=1}^{n} z_{it}, & \forall t \in \mathcal{T}_L, \quad & (2.26)\\
& \sum_{k=1}^{K} c_{kt} = u_t, & \forall t \in \mathcal{T}_L, \quad & (2.27)
\end{aligned}$$

node m splits on feature j

$$\begin{aligned}
& \mathbf{a}_m^\top \mathbf{x}_i \geq b_m - (1 - z_{it}), & i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \forall m \in A_R(t), \quad & (2.28)\\
& \mathbf{a}_m^\top (\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), & i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \forall m \in A_L(t), \quad & (2.29)
\end{aligned}$$

threshold of node m

$$\begin{aligned}
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, & i = 1, \ldots, n, \quad & (2.30)\\
& z_{it} \leq u_t, & \forall t \in \mathcal{T}_L, \quad & (2.31)\\
& \sum_{i=1}^{n} z_{it} \geq N_{\min} u_t, & \forall t \in \mathcal{T}_L, \quad & (2.32)\\
& \sum_{j=1}^{p} a_{jm} = d_m, & \forall m \in \mathcal{T}_B, \quad & (2.33)\\
& 0 \leq b_m \leq d_m, & \forall m \in \mathcal{T}_B, \quad & (2.34)\\
& d_m \leq d_{p(m)}, & \forall m \in \mathcal{T}_B \setminus \{1\}, \quad & (2.35)\\
& z_{it} \in \{0, 1\}, & i = 1, \ldots, n, \forall t \in \mathcal{T}_L, \quad & (2.36)\\
& l_t \in \{0, 1\}, & \forall t \in \mathcal{T}_L, \quad & (2.37)\\
& c_{kt} \in \{0, 1\}, & k = 1, \ldots, K, \forall t \in \mathcal{T}_L, \quad & (2.38)\\
& a_{jm} \in \{0, 1\}, & j = 1, \ldots, p, \forall m \in \mathcal{T}_B, \quad & (2.39)\\
& d_m \in \{0, 1\}, & \forall m \in \mathcal{T}_B. \quad & (2.40)
\end{aligned}$$

Binary Optimal Classification Trees (Bin-OCT)

A downside of the OCT formulation is that it creates multiple constraints for each sample in the dataset. Since the runtime of MILP solving grows worst-case exponentially in terms of the input size, this has a significant impact on the runtime. To enable the training of optimal classification trees on datasets with more samples, Verwer and Zhang proposed Bin-OCT [36]. The idea behind Bin-OCT is to use the fact that many samples in a dataset share feature values to combine multiple constraints into one. The name Bin-OCT refers to the use of binary variables to represent feature, threshold, and class label decisions.

The threshold values are also encoded using binary numbers, e.g. the threshold values 1, 2, 3, and 4 within some feature might be encoded by only 2 binary values as 00, 01, 10, 11 respectively. The notation is explained in Table 2.1.

Formulation The complete formulation for Bin-OCT is given below[4]:

$$\min \sum_{t\in\mathcal{T}_L}\sum_{k\in K} e_{t,k} \tag{2.41}$$

$$\text{s.t.} \quad \sum_{j\in J} \overset{\text{node } m \text{ splits on feature } j}{a_{m,j}} = 1, \qquad \forall m \in \mathcal{T}_B \tag{2.42}$$

$$\sum_{k\in K} \overset{\text{leaf } t \text{ predicts class } k}{c_{t,k}} = 1, \qquad \forall t \in \mathcal{T}_L, \tag{2.43}$$

$$\sum_{t\in\mathcal{T}_L} z_{it} = 1, \qquad i = 1,\dots,n, \tag{2.44}$$

$$M\, a_{m,j} + \sum_{\substack{i\in lr(\beta)\\ t\in ll(m)}} z_{i,t} + \sum_{\tau\in tl(\beta)} M(\overset{\text{node } m \text{ uses threshold } \tau}{b_{m,\tau}} - 1) \le M, \qquad \forall j\in J, \beta\in bin(j), i = 1,\dots,n, \tag{2.45}$$

$$M'\, a_{m,j} + \sum_{\substack{i\in ur(\beta)\\ t\in rl(m)}} z_{i,t} - \sum_{\tau\in tl(\beta)} M'\, b_{m,\tau} \le M', \qquad \forall j\in J, \beta\in bin(j), i = 1,\dots,n, \tag{2.46}$$

$$M''\, a_{m,j} + \sum_{\substack{i:X_{ij}>\tau^j_{\max}\\ t\in ll(m)}} z_{i,t} + \sum_{\substack{i:X_{ij}>\tau^j_{\min}\\ t\in rl(m)}} z_{i,t} \le M'', \qquad \forall m\in\mathcal{T}_B, j\in J, \tag{2.47}$$

$$\sum_{i:y_i=k} zi,t - M'''\, c_{t,k} \le e_{t,k}, \qquad \forall t\in\mathcal{T}_L, k\in K, \tag{2.48}$$

$$a_{m,j} \in \{0,1\}, \qquad \forall m\in\mathcal{T}_L, j\in J \tag{2.49}$$

$$b_{m,\tau} \in \{0,1\}, \qquad \forall m\in\mathcal{T}_L, \tau\in\mathrm{T} \tag{2.50}$$

$$c_{t,k} \in \{0,1\}, \qquad \forall t\in\mathcal{T}_L, k\in K \tag{2.51}$$

$$0 \le e_{t,k} \le \sum_{i=1}^{n} \mathbb{I}[y_i = k], \qquad \forall t\in\mathcal{T}_L, \forall k\in K, \tag{2.52}$$

$$0 \le z_{i,t} \le 1, \qquad \forall t\in\mathcal{T}_L, i = 1,\dots,n. \tag{2.53}$$

Limitations The motivation for Bin-OCT is to find good decision trees more quickly than previous methods for optimal classification trees. While the technique of combining multiple constraints with big-M values works for this goal (since fewer constraints speed up operations inside the solver), it does reduce the tightness of the LP relaxation. This reduces the solver's efficiency when proving optimality.

[4]There are some minor differences between the Bin-OCT formulation as presented in this dissertation and the formulation in [36]. The notation has been updated to be consistent with the other MILP methods in this chapter.

Bin-OCT proposes a pre-processing technique that removes thresholds that could only lead to suboptimal solutions. The idea is that when multiple samples in a sorted feature have the same label, it is not necessary to consider thresholds between these samples. However, Lin et al. [37] showed that this technique of 'bucketization' can remove optimal thresholds on specific datasets. Therefore when the goal is to train optimal classification trees, the bucketization pre-processing technique should not be applied.

Network Flow-Based Optimal Classification Trees (Flow-OCT)

In an effort to improve the tightness of MILP formulations for optimal classification trees, Aghaei et al. propose FlowOCT [35]. By formulating optimal classification trees as a network-flow problem, the authors create MILP instances that do not rely on big-M formulations. The method is relatively flexible and can be extended to optimize fairness [38] and robustness [39].

Formulation

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n} \sum_{t \in \mathcal{T}_L} f_{i,t,\text{sink}} && && (2.54)\\
\text{s.t.} \quad & \sum_{j \in J} \overset{\text{node } m \text{ uses binary feature } j}{a_{m,j}} = 1, && \forall m \in \mathcal{T}_B, && (2.55)\\
& \sum_{k \in K} \overset{\text{leaf } t \text{ predicts class } k}{c_{t,k}} = 1, && \forall t \in \mathcal{T}_L, && (2.56)\\
& f_{i,a(m),m} = f_{i,m,l(m)} + f_{i,m,r(m)}, && m \in \mathcal{T}_B, i = 1,\dots,n, && (2.57)\\
& f_{i,a(t),t} = f_{i,t,\text{sink}}, && t \in \mathcal{T}_L, i = 1,\dots,n, && (2.58)\\
& f_{i,\text{source},1} \leq 1, && i = 1,\dots,n, && (2.59)\\
& f_{i,m,l(m)} \leq \sum_{j \in J : X_{ij}=0} a_{m,j}, && m \in \mathcal{T}_B, i = 1,\dots,n, && (2.60)\\
& f_{i,m,r(m)} \leq \sum_{j \in J : X_{ij}=1} a_{m,j}, && m \in \mathcal{T}_B, i = 1,\dots,n, && (2.61)\\
& f_{i,t,\text{sink}} \leq c_{t,y_i}, && t \in \mathcal{T}_L, i = 1,\dots,n, && (2.62)\\
& a_{m,j} \in \{0,1\}, && \forall m \in \mathcal{T}_B, j \in J, && (2.63)\\
& c_{t,k} \in \{0,1\}, && \forall m \in \mathcal{T}_L, k \in K, && (2.64)\\
& f_{i,a(m),m} \in \{0,1\}, && \forall m \in \mathcal{T}, i = 1,\dots,n, && (2.65)\\
& f_{i,t,\text{sink}} \in \{0,1\}, && \forall t \in \mathcal{T}_L, i = 1,\dots,n. && (2.66)
\end{aligned}
$$

2.4.4 New General Formulation

We also propose a new formulation to generalize the previous MILP formulations to arbitrary classification loss functions (such as those used for algorithm selection and prescription). It only uses a binary variable for each possible split and one for each possible leaf prediction. Conceptually, it is similar to FlowOCT but does not require an explicit representation of the flow using binary variables. Instead, it expresses the path of a sample to a leaf directly in a single constraint.

Formulation

$$
\begin{aligned}
\min \quad & \frac{1}{n}\sum_{i=1}^{n} l_i && && (2.67)\\
\text{s.t.} \quad & \sum_{j\in J} \overset{\text{node } m \text{ uses binary feature } j}{a_{m,j}} = 1, && \forall m \in \mathcal{T}_B, && (2.68)\\
& \sum_{k\in K} \overset{\text{leaf } t \text{ predicts class } k}{c_{t,k}} = 1, && \forall t \in \mathcal{T}_L, && (2.69)\\
& l_i \geq \sum_{k} c_{t,k}\, L_i(k) - (L_i^{\max} - L_i^{\min})\Bigg(&& &&\\
& \quad \sum_{m\in A_L(t)} \sum_{j\in J\,:\,X_{ij}=1} a_{m,j} && i = 1,\ldots,n, \forall t \in \mathcal{T}_L, && (2.70)\\
& \quad + \sum_{m\in A_R(t)} \sum_{j\in J\,:\,X_{ij}=0} a_{m,j} - |A(t)|\Bigg), && &&\\
& a_{m,j} \in \{0,1\}, && \forall m \in \mathcal{T}_B, \forall j \in J, && (2.71)\\
& c_{t,k} \in \{0,1\}, && \forall t \in \mathcal{T}_L, \forall k \in K, && (2.72)\\
& L_i^{\min} \leq l_i \leq L_i^{\max}, && i = 1,\ldots,n. && (2.73)
\end{aligned}
$$

Flexibility of the General Formulation

This formulation is convenient because it generalizes multiple concepts while still being concise and relatively efficient:

- Instead of encoding only branch nodes of the form 'feature value $\leq$ threshold', it uses binary features to encode arbitrary predicates (like FlowOCT).
- It is not restricted to the $L_{0\text{-}1}$ (error rate) classification loss. Instead, it relies on an arbitrary loss function $L_i(k)$ that returns the loss associated with predicting class k for sample i. This allows for easy extensions to weighted classification or causal prescription.
- It introduces only 3 groups of decision variables and uses fewer total variables. Using fewer variables often reduces the number of branch-and-bound iterations required for solving.
- It uses only 3 groups of constraints: nodes choose one predicate, leaves choose one class, and if a sample lands in a leaf, its loss is forced.

- It does not rely on combined constraints to limit formulation size. Therefore it has a relatively tight relaxation.

The formulation can be applied to the $L_{0\text{-}1}$ loss, for example. In that case, we simply set $L_i^{\min} = 0$ and $L_i^{\max} = 1$, and substitute $L_i(k)$ for $\mathbb{I}[y_i \neq k]$. This replaces Constraint 2.70 by:

$$l_i \geq \sum_k \mathbb{I}[y_i \neq k] c_{t,k} - \left(\sum_{\substack{m \in A_L(t) \\ j \in J \,:\, X_{ij}=1}} a_{m,j} + \sum_{\substack{m \in A_R(t) \\ j \in J \,:\, X_{ij}=0}} a_{m,j} - |A(t)| \right). \tag{2.74}$$

2.4.5 Comparison

When developing algorithms for combinatorial optimization problems, the most specialized method is often the most efficient. For classification trees (and weighted classification variants), the dynamic programming-based methods are the most specialized. They make use of the structure in the search space of decision trees to reduce the amount of required computation. This means their time complexity scales linearly in terms of the number of samples and only exponentially in terms of features and tree depth. This is difficult to beat with integer programming and boolean satisfiability-based techniques since these general solvers run in exponential time in terms of samples, features, and tree size.

When training optimal classification trees in practice, we recommend using methods such as Streed [40]. If proven optimality is not required, it can be helpful to use a solver-based technique, as these can usually be stopped at any time to recover an approximate solution. The general MILP formulation is a flexible option for this. The remainder of this dissertation does not propose dynamic programming formulations for decision tree learning for another reason: current dynamic programming techniques require the optimization process of subtrees to be mostly separated, and this does not hold for the kinds of problems we will consider (robustness and sequential decision making).

Symbol	Type	Definition
$a_{j,m}$	variable	node m splits on feature j
b_m	variable	node m's continuous threshold value
$b_{m,\tau}$	variable	node m uses threshold τ
$c_{t,k}$	variable	leaf node t predicts class k
d_m	variable	node m is used
$e_{t,k}$	variable	number of samples with class k misclassified in leaf t
$f_{i,m,m'}$	variable	flow on the edge between m and m'
l_i	variable	loss for sample i
u_t	variable	leaf t is used
$z_{i,t}$	variable	sample i is in leaf t
X_{ij}	constant	value of data row i in feature j
y_i	constant	class label of data row i
n	constant	number of samples
α	constant	regularization term in OCT's objective
$N_{\min}$	constant	minimum number of samples required for creating a leaf node
n	constant	number of samples
D	constant	maximum depth of the decision tree
J	set	all features
K	set	all classes
$A(t)$	set	ancestors of node t
$A_L(t)$	set	... with left branch on the path to t
$A_R(t)$	set	... with right branch on the path to t
$\mathcal{T}_B$	set	all decision nodes
$\mathcal{T}_L$	set	all leaf nodes
$bin(j)$	set	feature j's binary encoding ranges
$lr(\beta)$	set	rows with values in β's lower range, $\beta \in bin(j)$
$ur(\beta)$	set	rows with values in β's upper range, $\beta \in bin(j)$
$tl(\beta)$	set	$b_{m,\tau}$ variables for β's ranges
$ll(m)$	set	node m's leaves under the left branch
$rl(m)$	set	node m's leaves under the right branch

Table 2.1: Summary of the notation used in the MILP formulations of this section.

2.5 Interpretability and Explainability

It makes sense that the ability to understand a machine learning model is a desired trait. However, in the literature, we currently find a variety of terms used to describe this property such as 'interpretable', 'explainable', 'XAI' (eXplainable Artificial Intelligence [41]), and 'transparent'. We briefly introduce the notions of transparency used in our work. In this dissertation, we will only use the terms *interpretable* and *explainable* and define similarly to [42]:

Interpretability. Or: *inherent* interpretability, refers to the property of a model that a human can directly understand how it makes predictions.

Explainability. The general concept of improving human understanding of a model, usually done by generating high-level explanations of predictions.

These definitions are abstract and can become clearer when matched with examples. For instance, small decision trees and sparse linear models are often referred to as **interpretable** since a user could, in theory, print these out on paper and understand exactly how they make predictions. Methods for **explainable** machine learning often rely on post-hoc explanation methods that use different techniques to highlight important features for the prediction. Although such explanations can help users understand the models, the important difference between interpretability and explainability is that model explanations do not necessarily allow the user to understand the complete model exactly. Therefore some works warn that explanations could mislead users by giving a false sense of model understanding [42–44].

2.5.1 Inherent Interpretability

Some types of models, such as decision trees, linear models, and rule lists, are generally considered inherently interpretable. However, their size or level of sparsity is important. In the mythos of model interpretability [45], Lipton separates the goals of interpretability for transparency into three levels: simulatability (understanding the whole model), decomposability (understanding individual components of the model), and algorithmic transparency (understanding the training algorithm). In this work, we mainly focus on simulatability, being able to understand the complete model, and for this, it is vital that the model is reasonably simple. 'Reasonably' is a subjective term here and cannot generally be defined. We can, however, give some examples of models that are often considered inherently interpretable but are not reasonably simple. One example is a decision tree with more than 50 nodes. Such a model is too large to succinctly visualize, and humans cannot easily comprehend their predictions. Similarly, a linear model using more than 50 non-zero coefficients will be difficult to understand: making predictions by hand will take minutes. In this dissertation, we usually resort to training trees of at most depth 3 or 4 (at most 8 or 16 branch nodes, respectively). Although larger sizes could increase predictive performance while still being interpreted, we choose these to stay on the safe side of the subjective 'reasonably simple' requirement.

2.5.2 Feature Importance

Although the main focus of this dissertation is on inherent interpretability, we highlight tree ensemble feature importance, which is a commonly used post-hoc explanation method. Most post-hoc machine learning explanations are based on feature attribution. In feature attribution explanations, some method is used to determine how important each of the features of some input is for the final prediction. For most machine learning models, feature importances are estimated by some model agnostic methods such as SHAP [46] and LIME [47]. However, for tree-based models, there are techniques that use information inside the model to determine feature importance more efficiently [48]. Since in typical decision trees, each node selects one feature to split on, and these nodes can be easily enumerated, this provides useful properties for generating explanations.

Misleading Explanations Although feature importance explanations can be useful, it is important to consider their limitations. There are situations where feature importances are not truthful to the underlying models or can be misleading. For example, when using feature importances generated from tree ensembles, we have to consider the number of unique values that the features have. When a feature has many more unique values than others, and thus allows the tree to split on that feature more often, there is a high probability of the model using more nodes that split on that feature. This does not mean that the feature holds more information than another one. Another problem occurs when multiple features are correlated. In this case, the decision tree gains similar information when splitting on any of the features, and thus, importance will be arbitrarily spread over the features. Lastly, model-agnostic explanation methods such as LIME and SHAP are not robust [44, 49]. By carefully making small changes to the inputs, they can be fooled into generating different explanations.

Bibliography

[1] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees,* (1984).

[2] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, *Scikit-learn: Machine learning in python,* the Journal of machine Learning research **12**, 2825 (2011).

[4] P. Bartlett, Y. Freund, W. S. Lee, and R. E. Schapire, *Boosting the margin: A new explanation for the effectiveness of voting methods,* The annals of statistics **26**, 1651 (1998).

[5] L. E. Raileanu and K. Stoffel, *Theoretical comparison between the gini index and information gain criteria,* Annals of Mathematics and Artificial Intelligence **41**, 77 (2004).

[6] A. Perez-Lebel, G. Varoquaux, M. Le Morvan, J. Josse, and J.-B. Poline, *Benchmarking missing-values approaches for predictive models on health databases,* GigaScience **11**, giac013 (2022), https://academic.oup.com/gigascience/article-pdf/doi/10.1093/gigascience/giac013/43384549/giac013.pdf .

[7] D. Heath, S. Kasif, and S. Salzberg, *Induction of oblique decision trees,* in *IJCAI*, Vol. 1993 (Citeseer, 1993) pp. 1002–1007.

[8] S. K. Murthy, S. Kasif, and S. Salzberg, *A system for induction of oblique decision trees,* Journal of artificial intelligence research **2**, 1 (1994).

[9] D. Bertsimas and J. Dunn, *Optimal classification trees,* Machine Learning **106**, 1039 (2017).

[10] C. Aytekin, *Neural networks are decision trees,* arXiv preprint arXiv:2210.05189 (2022).

[11] L. Grinsztajn, E. Oyallon, and G. Varoquaux, *Why do tree-based models still outperform deep learning on typical tabular data?* in *Advances in Neural Information Processing Systems*, Vol. 35, edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Curran Associates, Inc., 2022) pp. 507–520.

[12] L. Breiman, *Random forests,* Machine Learning **45**, 5 (2001).

[13] L. Breiman, *Bagging predictors,* Machine learning **24**, 123 (1996).

[14] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Vol. 2 (Springer, 2009).

[15] A. Curth, A. Jeffares, and M. van der Schaar, *Why do random forests work? understanding tree ensembles as self-regularizing adaptive smoothers,* arXiv preprint arXiv:2402.01502 (2024).

[16] J. H. Friedman, *Greedy function approximation: a gradient boosting machine,* Annals of statistics , 1189 (2001).

[17] J. H. Friedman, *Stochastic gradient boosting,* Computational statistics & data analysis **38**, 367 (2002).

[18] L. Mason, J. Baxter, P. Bartlett, and M. Frean, *Boosting algorithms as gradient descent,* Advances in neural information processing systems **12** (1999).

[19] T. Chen and C. Guestrin, *Xgboost: A scalable tree boosting system,* in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, *Lightgbm: A highly efficient gradient boosting decision tree,* Advances in neural information processing systems **30** (2017).

[21] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, *Catboost: unbiased boosting with categorical features,* Advances in neural information processing systems **31** (2018).

[22] Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting,* Journal of Computer and System Sciences **55**, 119 (1997).

[23] P. Geurts, D. Ernst, and L. Wehenkel, *Extremely randomized trees,* Machine learning **63**, 3 (2006).

[24] F. T. Liu, K. M. Ting, and Z.-H. Zhou, *Isolation forest,* in *2008 eighth ieee international conference on data mining* (IEEE, 2008) pp. 413–422.

[25] S. Hariri, M. C. Kind, and R. J. Brunner, *Extended isolation forest,* IEEE transactions on knowledge and data engineering **33**, 1479 (2019).

[26] S. C. Tan, K. M. Ting, and T. F. Liu, *Fast anomaly detection for streaming data,* in *Twenty-second international joint conference on artificial intelligence* (Citeseer, 2011).

[27] G. B. Dantzig, *Origins of the simplex method,* in *A history of scientific computing* (1990) pp. 141–151.

[28] N. Karmarkar, *A new polynomial-time algorithm for linear programming,* in *Proceedings of the sixteenth annual ACM symposium on Theory of computing* (1984) pp. 302–311.

[29] A. Land and A. Doig, *An automatic method of solving discrete programming problems,* Econometrica **28**, 497 (1960).

[30] R. Gomory and R. C. S. M. CA, *An Algorithm for the Mixed Integer Problem: Notes Linear Programming and Extensions-Part 54* (Rand, 1960).

[31] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj, *Gomory cuts revisited,* Operations Research Letters **19**, 1 (1996).

[32] L. Hyafil and R. L. Rivest, *Constructing optimal binary decision trees is np-complete,* Information Processing Letters **5**, 15 (1976).

[33] R. M. Karp, *Reducibility among combinatorial problems* (Springer, 2010).

[34] M. R. Garey, D. S. Johnson, *et al.*, *A guide to the theory of np-completeness,* Computers and intractability , 37 (1990).

[35] S. Aghaei, A. Gómez, and P. Vayanos, *Strong optimal classification trees,* arXiv preprint arXiv:2103.15965 (2021).

[36] S. Verwer and Y. Zhang, *Learning optimal classification trees using a binary linear program formulation,* in *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33 (2019) pp. 1625–1632.

[37] J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer, *Generalized and scalable optimal sparse decision trees,* in *International Conference on Machine Learning* (PMLR, 2020) pp. 6150–6160.

[38] N. Jo, S. Aghaei, J. Benson, A. Gomez, and P. Vayanos, *Learning optimal fair decision trees: Trade-offs between interpretability, fairness, and accuracy,* in *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society* (2023) pp. 181–192.

[39] N. Justin, S. Aghaei, A. Gómez, and P. Vayanos, *Learning optimal classification trees robust to distribution shifts,* arXiv preprint arXiv:2310.17772 (2023).

[40] J. van der Linden, M. de Weerdt, and E. Demirović, *Necessary and sufficient conditions for optimal decision trees using dynamic programming,* in *Advances in Neural Information Processing Systems*, Vol. 36, edited by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Curran Associates, Inc., 2023) pp. 9173–9212.

[41] D. Gunning, *Darpa's explainable artificial intelligence (xai) program,* in *Proceedings of the 24th International Conference on Intelligent User Interfaces*, IUI '19 (Association for Computing Machinery, New York, NY, USA, 2019) p. ii.

[42] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,* Nature machine intelligence **1**, 206 (2019).

[43] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim, *Sanity checks for saliency maps,* Advances in neural information processing systems **31** (2018).

[44] A. Nadeem, D. Vos, C. Cao, L. Pajola, S. Dieck, R. Baumgartner, and S. Verwer, *Sok: Explainable machine learning for computer security applications,* in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)* (IEEE, 2023) pp. 221–240.

[45] Z. C. Lipton, *The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery.* Queue **16**, 31 (2018).

[46] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions,* in *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017) pp. 4765–4774.

[47] M. T. Ribeiro, S. Singh, and C. Guestrin, *" why should i trust you?" explaining the predictions of any classifier,* in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016) pp. 1135–1144.

[48] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, *From local explanations to global understanding with explainable ai for trees,* Nature Machine Intelligence **2**, 2522 (2020).

[49] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, *Fooling lime and shap: Adversarial attacks on post hoc explanation methods,* in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (2020) pp. 180–186.

Optimizing Decision Trees for **Adversarial Robustness**

3 Introduction to Robust Decision Trees

While regular decision trees and decision tree ensembles perform well at typical supervised learning tasks, their performance degrades when there is uncertainty in the data. In fact, when making small changes to regular decision tree inputs, it is possible to change the predictions that the inputs receive significantly. Being susceptible to these adversarial perturbations means that the model is not adversarially robust. In this part, we will consider the problem of optimizing decision tree models for adversarial robustness by considering uncertainty during the training phase. Specifically, we will consider that an adversary can make small changes to the data points in order to cause misclassification. An attractive property of robust decision trees and decision tree ensembles is that their robustness can be tractably verified, whereas more complex models such as neural networks typically only heuristically improve robustness.

In this chapter, we introduce adversarial robustness within the field of machine learning. We will start by defining robustness and show why robustness is an important property. Next, we will discuss different methods of evaluating the robustness of machine learning models and, more specifically, (ensembled) decision trees. We introduce general techniques that consider worst-case adversarial perturbations during the optimization process. Lastly, we define poisoning robustness as a special kind of robustness that considers the ability of potential attackers to add or remove samples from the training data of a machine learning model.

3.1 Adversarial Robustness

In 2013, Szegedy et al. [1] found that making very small changes to inputs of neural networks could cause the model to misclassify them completely. These perturbed inputs were so similar to their unperturbed versions that humans could not notice a difference. Such perturbed inputs that cause models to display unintended behaviors are referred to as adversarial examples. The work by Szegedy et al. and earlier works by Biggio et al. [2] that demonstrated the fragility of machine learning models to modified inputs motivated research into optimizing machine learning models that are robust to adversarial examples.

For a model to be robust to adversarial examples, we want to ensure that there is no 'small' change that one could make to an input such that it gets misclassified by the model. This is often defined by a set of user-specified perturbations Δ, each of which should not cause a misclassification when added to an input x. So for a classifier C and dataset D we aim to satisfy:

$$C(x+\delta) = y, \quad \forall (x,y) \in D, \forall \delta \in \Delta. \tag{3.1}$$

For mathematical convenience the set Δ is often described by a L^p-norm ball of radius ϵ: $\Delta = \{\delta : ||\delta||_p \leq \epsilon\}$, for some $p \in [1,\infty]$. The L^∞ norm is particularly convenient for decision trees since their axis-aligned splits partition the space into high-dimensional boxes just like an L^∞ ball of fixed radius. However, while mathematically convenient, realistic perturbations are often not accurately modeled by L^p norms.

Decision trees can also suffer from adversarial examples. In Figure 3.1 we visualize a dataset and its associated decision tree. While the decision tree achieves an accuracy of 80% by using just one branching node, the accuracy gets reduced to 0% when applying small adversarial perturbations to each sample. We want to find decision trees that are robust to small changes in the data. We can define 'small changes in the data' as, for example, changes within a box with a fixed radius around each sample. In that case we hope to find a tree such as the one visualized in Figure 3.2. This robust decision tree achieves a reduced accuracy of 70% but maintains that accuracy after adversarial perturbations.

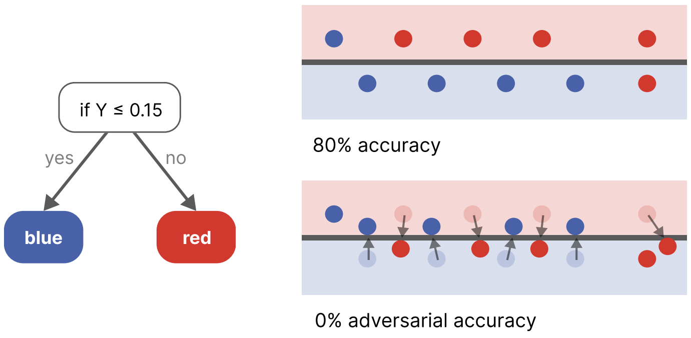


Figure 3.1: A **non-robust** depth-1 decision tree and its predictions on a toy dataset. While the tree has good associated accuracy (80%), its accuracy when accounting for small adversarial perturbations is 0%. Based on the example by Chen et al. [3].

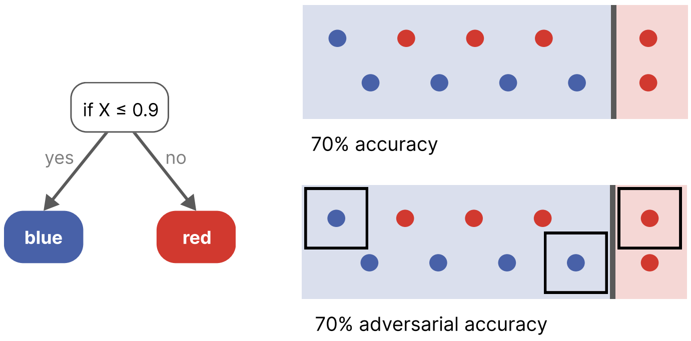


Figure 3.2: A **robust** depth-1 decision tree and its predictions on a toy dataset. While the tree has a reduced accuracy of 70%, its accuracy when accounting for adversarial perturbations limited by a box of radius 0.1 is still 70%. Based on the example by Chen et al. [3].

3.2 Evaluating Robustness

To evaluate the adversarial robustness of a machine learning model, we are usually interested in answering one of two questions:

Definition 1. *The **feasible** adversarial example problem: Find an adversarial example $x+\delta$ for input x with label y such that $C(x+\delta) \neq y$ and the perturbation δ is in the set of possible perturbations Δ.*

Definition 2. *The **optimal** adversarial example problem: Find an adversarial example $x+\delta$ for input x with label y such that $C(x+\delta) \neq y$, the perturbation δ is in the set of possible perturbations Δ, **and minimize** $d(x, x+\delta)$ **for some distance function** d.*

Given a solution to the optimization version of the problem, the feasibility problem is trivial to answer.

	Distance norm		
Model class	L^0	$L^p, p \in (0,\infty)$	L^∞
decision tree	$\mathcal{O}(\mathcal{T})$	$\mathcal{O}(\mathcal{T})$	$\mathcal{O}(\mathcal{T})$
stump ensemble	$\mathcal{O}(\mathcal{T}\log\mathcal{T})$	$\mathcal{O}(2^{\mathcal{T}})$	$\mathcal{O}(\mathcal{T}\log\mathcal{T})$
tree ensemble	$\mathcal{O}(2^{\mathcal{T}})$	$\mathcal{O}(2^{\mathcal{T}})$	$\mathcal{O}(2^{\mathcal{T}})$

Table 3.1: Robustness verification time complexities (based on [4]) for decision tree-based models. For all L^p-norms, single decision trees can be verified in linear time, while (additive) tree ensembles are NP-hard to verify. For stump ensembles, special algorithms for the L^0 and L^∞ norms allow for linearithmic runtimes.

However, depending on the model, it can be computationally challenging to evaluate its robustness. For example, due to the scale and structure of typical neural networks, determining whether an adversarial example within Δ is often intractable. Therefore, for neural networks, we commonly apply heuristic methods to find adversarial examples and use approximated bounds to prove the non-existence of adversarial examples. A major advantage of decision trees and decision tree ensembles over neural networks is that they usually allow for tractable computation of adversarial examples. When we evaluate a model's robustness, we assume we have access to its parameters, which in the literature is referred to as transparent or white-box attacking.

Wang et al. [4] collected and proved the time complexities for different kinds of tree-based models. Their results are summarized in Table 3.1. In the sections below, we explain how to derive the complexities of two important cases: single decision trees and tree ensembles.

3.2.1 Adversarial Examples for Decision Trees

Robustness against L^p norm perturbations can be verified in linear time for single decision trees. To verify whether there exists an adversarial perturbation δ such that $\|\delta\|_p \leq \epsilon$ and $C(x+\delta)$ is a misclassification the idea is to simply compute the minimal perturbation required to move point x into every leaf of the tree. Given that the minimal perturbation for a point to a leaf can be computed quickly (linear in the number of dimensions), the complete procedure runs in time linear in terms of the number of leaves.

Minimal perturbation onto a leaf The minimum norm perturbation δ^* required to move a point x into a decision tree leaf can be computed quickly in closed form. This is possible because decision trees with axis-aligned splits create box-shaped leaves that allow us to find a minimum perturbation in each dimension separately. Denote the box-shaped space that a leaf covers as the set $B = \{x : l_j^t \leq x_j \leq u_j^t \; \forall j\}$, i.e. a box defined by a lower bound l_j and upper bound u_j for each feature j. For any L^p norm the minimum perturbation δ^* is:

$$\delta^* := \arg\min_{x' \in B} \|x - x'\|. \tag{3.2}$$

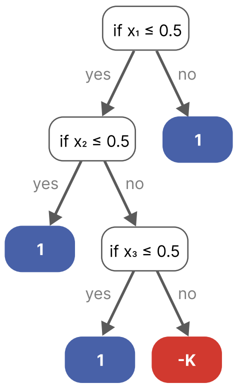


Figure 3.3: Decision tree created for clause $(x_1 \vee \neg x_2 \vee \neg x_3)$ in the reduction from 3-SAT to adversarial example finding. Finding an adversarial example for an ensemble of these trees constitutes solving a 3-SAT formula, proving that finding adversarial examples in tree ensembles is NP-hard.

Which can be computed in closed form by separating each feature j:

$$\delta_j^* = \begin{cases} x_j & l_j^t \leq x_j \leq u_j^t \\ l_j^t & x_j < l_j^t \\ u_j^t & x_j > u_j^t \end{cases} . \tag{3.3}$$

3.2.2 Adversarial Examples for Decision Tree Ensembles

Finding adversarial examples for decision tree ensembles is a non-trivial task. Below we repeat a short proof on the NP-hardness from Kantchelian et al. [5]. After, we explain the MILP formulation that can be used to solve this NP-hard problem to find optimal adversarial examples.

NP-hardness

For brevity, we only repeat the high-level idea of the NP-hardness proof for finding adversarial examples in tree ensembles. For details, we refer to the original paper [5]. The idea behind the proof is to reduce the NP-complete problem of 3-SAT to finding an adversarial example in a tree ensemble (Definition 1). This proves that finding adversarial examples is at least as difficult as solving 3-SAT.

Problem is in NP Without loss of generality, define the prediction function of a tree ensemble as $C(x) = \text{sign}(\sum_i T_i(x))$, i.e. predicting 1 when the sum of individual tree predictions is greater than 0, and -1 otherwise. Computing the prediction $C(x)$ for an instance x can be done in time linear in the size of the tree ensemble. Therefore, a solution to the problem can be verified in polynomial time, and the problem is in NP.

Problem is NP-hard Consider a 3-SAT instance S consisting of K clauses, each with 3 literals. We will construct a decision tree with 3 branch nodes for each clause.

Take, for example, the clause $(x_1 \lor \neg x_2 \lor \neg x_3)$; this clause will be transformed into the decision tree given in Figure 3.3. Each literal is encoded into a binary feature, for instance, x_1 is encoded as feature x_1. Each branch node of the tree will then test whether the value of x_j is true/false (1 or 0) using the predicate $[x_j \leq 0.5]$. Then the tree is constructed in such a way that any satisfying assignment to the variables in the clause causes the tree to 1, while any assignment that does not satisfy the clause results in a prediction of $-K$. Since a tree predicts at most 1, this means that if any clause is not satisfied, the sum of tree predictions can be at most $-K + \sum_{i=1}^{K-1} 1 = -K + K - 1 = -1$ (one tree predicts $-K$ while all remaining trees predict 1) which results in $C(x) = -1$. Whenever an input represents a solution to the 3-SAT instance that satisfies all clauses, the trees will all predict 1, which results in a positive prediction $C(x) = 1$. Therefore the presence/absence of an adversarial example $x + \delta$ causing $C(x + \delta) = 1$ directly translates to a solution to the 3-SAT instance S. Finding adversarial examples in tree ensembles is NP-hard. □

MILP Formulation

While the problem of finding adversarial examples in tree ensembles is NP-hard, this does not mean that solving the problem is necessarily intractable. Kantchelian et al. [5] proposed a MILP formulation that can be used to find adversarial examples for tree ensembles. Although for large random forests that contain deep trees, the MILP instances can take hours to solve, they generally run in seconds or minutes for models such as gradient boosting ensembles with much smaller trees.

The formulation is made for decision trees that use predicates of the type 'the value of feature $j \leq$ threshold b'. The idea of the formulation is not to explicitly search for a perturbation δ but to find out what predicates inside the ensemble must be simultaneously satisfied to cause a misclassification. The size of the perturbation that is required to satisfy these predicates is then cleverly encoded in the objective. We will first introduce the decision variables. We highlight them in bold for clarity. Define a decision variable $\mathbf{p_j^{(i)}} \in \{0, 1\}$ for every unique branch node in feature j, such that the predicates are ordered by threshold value with i, i.e. when predicate $\mathbf{p_j^{(i)}}$ is satisfied $\mathbf{p_j^{(i+1)}}$ is also satisfied (Constraint 3.5). Define a decision variable $\mathbf{l_{k,t}} \in \{0, 1\}$ that indicates whether or not the adversarial example lies in leaf node t of tree k. The adversarial example will be in exactly one leaf in every tree $k = 1, ..., T$ (Constraint 3.6).

To determine whether the perturbed sample can reach a certain leaf the variables $\mathbf{l_{k,t}}$ must be forced to 0 when a predicate disallows the perturbed sample to follow a path to that leaf. For this, we introduce the function true(t, j, i) that determines whether the ith predicate of feature j needs to be true to reach leaf t, and false(t, j, i) for whether this should be false. When a predicate $\mathbf{p_j^{(i)}}$ is true/false, we force the leaves on the respectively false/true sides to 0 with Constraints 3.7 and 3.8. Without loss of generality, the formulation is designed to optimize for a prediction of $C(x + \delta) = 1$. For an additive tree ensemble, this means that the sum of all the predicted values of the tree should be greater or equal to 0 (Constraint 3.9). We use the notation $v_{k,t}$ to denote the constant value of leaf t in tree k, I_j the number of unique threshold values for feature j, T the total number of trees, and $\mathcal{T}_L^k$ the indices t of the leaves for tree k.

$$\begin{aligned}
\min \quad & \sum_{j \in J} \sum_{i=1}^{I_j} w_{j,i} p_j^{(i)} && && (3.4)\\
\text{s.t.} \quad & p_j^{(i)} \leq p_j^{(i+1)}, && \forall j \in J, i = 1, \ldots, I_j - 1 && (3.5)\\
& \sum_{t \in \mathcal{T}_L^k} \mathbf{l_{k,t}} = 1, && k = 1, \ldots, T && (3.6)\\
& \sum_{t : t \in \mathcal{T}_L^k \wedge \text{true}(t,j,i)} \mathbf{l_{i,t}} \leq \mathbf{p_j^{(i)}}, && \forall j \in J, i = 1, \ldots, I_j, k = 1, \ldots, T && (3.7)\\
& 1 - \sum_{t : t \in \mathcal{T}_L^k \wedge \text{false}(t,j,i)} \mathbf{l_{i,t}} \geq \mathbf{p_j^{(i)}}, && \forall j \in J, i = 1, \ldots, I_j, k = 1, \ldots, T && (3.8)\\
& \sum_{k=1}^{T} \sum_{t \in \mathcal{T}_L^k} v_{k,t} \mathbf{l_{k,t}} \geq 0 && && (3.9)\\
& \mathbf{p_j^{(i)}} \in \{0, 1\} && \forall j \in J, i = 1, \ldots, I_j && (3.10)\\
& \mathbf{l_{k,t}} \in \{0, 1\} && \forall k = 1, \ldots, T, t \in \mathcal{T}_L^k. && (3.11)
\end{aligned}$$

Denote the perturbed sample as $x' = x + \delta$. The objective is to minimize the distance from the perturbed sample x' to the original sample x. Recall that an arbitrary L^p norm can be computed in n dimensions as:

$$\|x' - x\|_p = \left(\sum_{i=1}^{n} |x' - x|^p \right)^{\frac{1}{p}}.$$

To enable formulations for arbitrary L^p norms the distance that is minimized is raised to the power p:

$$\|x' - x\|_p^p = \sum_{i=1}^{n} |x' - x|^p.$$

So the objective is to minimize the sum of absolute differences raised to the power p in every dimension. These values can be precomputed for every interval indicated by the pairs $(p_j^{(i)}, p_j^{(i+1)})$ and used to set the values of $w_{j,i}$ in the objective.

Instead of finding a minimal adversarial example, it is also common to use the formulation to verify whether an adversarial example exists within a certain distance. In this case, the objective is removed and turned into a constraint. This feasibility version of the formulation runs significantly faster on average.

3.3 Robust Optimization

Recall from chapter 2 that we can define learning as an optimization problem by minimizing a loss function L. In this typical learning setting, we aim to find model parameters θ that result in the minimal expected loss on the training data:

$$\min_{\theta} \mathbb{E}_{x,y} L(\theta, x, y). \tag{3.12}$$

In the robust (or adversarial) setting, we will take into account that the samples x_i could be perturbed by an adversary. Define the set Δ that contains all possible perturbations. The robust learning problem (following the definition of Madry et al. [6]) is then defined as:

$$\min_{\theta} \mathbb{E}_{x,y} \left[\max_{\delta \in \Delta} L(\theta, x + \delta, y) \right]. \tag{3.13}$$

Before computing the loss of model parameters θ on some pair (x, y), the adversary finds the worst-case perturbation δ and applies it to x. These min-max optimization problems are common in the field of robust optimization and are often significantly more challenging to solve than purely minimization or maximization problems.

A common approach to solving min-max problems is to break the problem up into two parts. First, one computes some new values of θ and fixes these. Then, since the values of θ are fixed constants, solve the pure maximization problem

$$\max_{\delta \in \Delta} L(\theta, x + \delta, y) \tag{3.14}$$

to find the worst-case perturbation. This technique is used in adversarial learning for neural networks, for example [6]. One of our contributions in this dissertation is a MILP reformulation of the robust learning problem for decision trees in such a way that the min-max problem effectively becomes a pure minimization problem (Chapter 5). This way, the problem can be directly solved without having to resort to solving it in two separate stages.

3.3.1 Epigraph reformulations

Within the field of mathematical programming, there is a general way in which minimax problems can be naturally split into two stages. Consider an optimization problem:

$$\min_{x \in X} \max_{u \in U} f(x, u). \tag{3.15}$$

If U is a finite set, we can rewrite the problem into its epigraph form by introducing a constraint for each possible uncertain value u:

$$\begin{aligned} \min \quad & r && (3.16) \\ \text{s.t.} \quad & r \geq f(x, y), \quad \forall u \in U. && (3.17) \end{aligned}$$

When the constraints 3.17 can be expressed using linear relations, this formulation is especially useful as the problem can then be encoded as an (Integer) Linear Program. Of course, if the set of U is large, then the optimization problem will also have many constraints. Fortunately, for many specific optimization problems, it is not necessary to consider all these constraints to solve the problem; instead, one can use *constraint generation* techniques. In constraint generation, the idea is to iteratively add only those constraints that are needed to cut away infeasible solutions. This means that when the solver proposes a solution (r', x') that should be infeasible, one solves a problem like $u^* = \arg\max_{u \in U} f(x', u)$ to add the constraint $r' \geq f(x', u^*)$ which invalidates the previous solution. In modern ILP solvers such as GUROBI[1], this procedure can be efficiently implemented in the search using callbacks. Such an implementation could be used to extend the methods from Chapter 5 to different kinds of perturbation sets.

[1]https://www.gurobi.com/

3.3.2 Distributionally Robust Optimization

Robust optimization optimizes an objective for the worst-case outcome from some user-specified uncertainty set. This can be overly pessimistic since uncertainty in the real world is often not adversarial but stochastic. For example, if we consider robustness in a machine learning setting, the probability that all samples will be set to their worst possible locations when they are randomly perturbed is very low. A technique that bridges the gap between robust and stochastic optimization is distributionally robust optimization. Distributionally robust optimization defines the uncertainty set as a set of probability distributions over the uncertain entities in the problem. This means that in our example of machine learning robustness, an adversary does not perturb each sample individually but collectively perturbs the entire dataset. An example of an uncertainty set could be 'the sum of all samples' perturbation sizes should be within the total perturbation budget ϵ.'

The general distributionally robust optimization problem for machine learning can be written as:

$$\inf_{\theta} \sup_{\mathbb{Q}\in U(X,Y)} \mathbb{E}_{(x,y)\sim\mathbb{Q}}\left[L(\theta,x,y)\right], \tag{3.18}$$

where $U(X,Y)$ defines the uncertainty set around the dataset given by X and Y, and $\mathbb{Q}$ is the perturbed distribution. One minimizes the expected loss of the worst-case distribution shift from the original dataset. Under certain conditions, such a problem can be solved using constraint generation techniques based on the epigraph reformulations mentioned earlier. Some work already considers these kinds of approaches for decision trees [7]. Distributionally robust machine learning has promising use cases in, for example, generalization guarantees [8] and as a regularization technique [9].

3.4 Robust Decision Tree Learning

We summarize previous works that propose heuristic methods for optimizing robust decision trees. In Table 3.2 we compare each algorithm's runtime complexity and threat model. Most of these algorithms build on ideas used for popular greedy algorithms for classification and regression trees (CART).

Algorithm	Runtime	Threat model
GROOT	$\mathcal{O}(n\log n)$	L^∞ and variations
Provably robust boosting	$\mathcal{O}(n^2)$	L^∞
TREANT	$\mathcal{O}(n^2)$	axis-aligned rules
Chen et al. exact	$\mathcal{O}(n^2)$	L^∞
Chen et al. heuristic	$\mathcal{O}(n\log n)$	L^∞
MILP hardening	$\mathcal{O}(n\log n)$	L^0 / L^1 / L^2 / L^∞
Approx. hardening	$\mathcal{O}(n\log n)$	L^0

Table 3.2: Overview of algorithms for fitting robust decision trees. Runtimes in terms of n number of samples, all algorithms also grow linearly in number of features and exponentially in depth.

3.4.1 Robust Boosting of Tree Ensembles

Setting the foundations of robust decision trees, Kantchelian et al. [5] propose an approach to improve the robustness of tree ensembles and prove that finding adversarial examples under distance constraints is NP-hard for tree ensembles. Their robust ensemble learning method works by robust boosting, i.e., incrementally adding learners to the ensemble to compensate for the fragility of the preceding learners. They also provide a MILP formulation to solve the adversarial example optimization problem for arbitrary L_p norm, which we use to verify the robustness of our models in Section 4.5.

3.4.2 Robust Decision Trees

Chen et al. [3] present an algorithm that fits robust decision trees against L^∞ norm perturbations by using a new splitting criterion. This criterion is the worst-case information gain or Gini impurity when an attacker moves points within an L^∞ radius. The authors find that they can compute the criterion exactly using gradient descent which takes $\mathcal{O}(n)$ time in terms of n samples. They deem this computation intractable for boosting ensembles and, therefore, give a fast heuristic based on four representative cases. GROOT's criterion is equivalent to the exact criterion but speeds up the computation to $\mathcal{O}(1)$ time and thereby enables its use in ensembles.

3.4.3 TREANT

TREANT [10] introduces a more flexible approach to specifying attacker capabilities. By allowing the user to describe an adversary using axis-aligned rules, attackers can be more realistically modeled with asymmetric changes and different constraints for different axes. Also, attackers can be modeled with a 'budget' that they can spend on changing data points, which allows the user to evaluate robustness against attackers of different strengths. TREANT still greedily builds a tree, but it directly optimizes a loss function instead of using a splitting criterion. Although this allows TREANT to train against a variety of attackers, their algorithm deploys a solver to optimize the loss function and pre-computes all possible attacks which takes in the order of hours to run.

3.4.4 Provably Robust Boosting

Where Chen et al. and TREANT describe algorithms for fitting a single robust tree, provably robust boosting [11] directly fits a robust ensemble. The authors find that they can efficiently compute the adversarial loss for boosted decision stumps and use this to derive an upper bound on the adversarial loss of boosted decision trees. By optimizing this bound on boosted decision trees, they reach state-of-the-art performance on adversarial accuracy in tree ensembles and can compete with results from neural networks. While their ensembles contain many shallow trees which grants fast inference time, the training time of the method is in the order of hours.

3.5 Poisoning robustness

Data poisoning attacks are attacks in which a malicious actor adds, removes, or modifies the data that a machine learning model is trained on. By making specific changes to the dataset, attackers can, for example, reduce the model's performance or plant a backdoor that allows the attacker to inject a specific 'trigger pattern' into an input of the model at test time to control its prediction. While much work has gone into defending against poisoning attacks [12–14] and into provable robustness against other attacks such as evasion [15, 16] it is notoriously difficult to provide provable guarantees on poisoning defenses. In Chapter 7, we consider defending against data poisoning attacks using differential privacy.

3.5.1 Differential privacy

Differential privacy [17–19] provides strong privacy guarantees for algorithms over aggregate datasets, which implies that the existence of any record in the dataset does not influence the output probability with factors ϵ and δ. This property prevents membership attacks, attacks aimed at determining whether specific samples were included in the train set of a model, with a high probability if ϵ is chosen small enough.

Definition 3 (Differential privacy). *A randomized algorithm $\mathcal{M}$ satisfies (ϵ,δ)-differential privacy if for all neighboring datasets $\mathcal{D},\mathcal{D}' \in \mathbb{N}^{|\mathcal{X}|}$ differing in one element, and any $\mathcal{S} \subseteq$ Range($\mathcal{M}$),*

$$\Pr[\mathcal{M}(\mathcal{D}) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{M}(\mathcal{D}') \in \mathcal{S}] + \delta \tag{3.19}$$

where $\mathbb{N}$ is the set of non-negative integers and $\mathcal{X}$ is the universe for all datasets. If δ is 0, $\mathcal{M}$ satisfies ϵ-differential privacy.

Among differentially private mechanisms, the Laplace mechanism adds noise to a numerical output, and the exponential mechanism returns a precise output among a group according to the utility scores for each element. Both mechanisms are widely used and are defined as follows.

Definition 4 (Laplace mechanism [18]). *A randomized algorithm $\mathcal{M}$ satisfies ϵ-differential privacy over a real value query $f : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$ if*

$$\mathcal{M}(\mathcal{D},\ f,\ \epsilon) = f(\mathcal{D}) + (y_1\ \ldots\ y_k), \qquad y_i \sim \mathrm{Lap}\left(\frac{\Delta f}{\epsilon}\right) \tag{3.20}$$

where $\mathrm{Lap}(b)$ is the Laplace distribution with scale b that $\mathrm{Lap}(x \mid b) = \frac{1}{2b}\exp(-\frac{|x|}{b})$, and Δf is the l_1-sensitivity that

$$\Delta f = \max_{X,X' \in \mathbb{N}^{|\mathcal{X}|}, \|X-X'\|_1 \leq 1} \|f(X) - f(X')\|_1 \tag{3.21}$$

With the Laplace mechanism, the Laplace noise is added to the accurate output of the query f so that the output of $\mathcal{M}$ satisfies ϵ-differential privacy over the query. Other mechanisms, such as the Gaussian mechanism and geometric mechanism [20], achieve differential privacy in a similar way. The geometric mechanism is similar to the Laplace mechanism but works with integer values. An algorithm $\mathcal{M}(\mathcal{D},\ f,\ \epsilon)$ is ϵ-differentially private when the noise follows the two-sided geometric distribution:

$$\Pr[y_i = \delta] = \frac{1-\epsilon}{1+\epsilon}\epsilon^{|\delta|} \tag{3.22}$$

with a query f, the parameter $\epsilon \in (0,1)$ and every integer δ.

Definition 5 (Exponential mechanism [21]). *A randomized algorithm $\mathcal{M}$ satisfies ϵ-differential privacy over a utility function $u : \mathbb{N}^{|\mathcal{X}|} \times \mathcal{R} \rightarrow \mathbb{R}$ if $\mathcal{M}$ selects an element $t \in \mathcal{R}$ with the probability that*

$$\Pr[\mathcal{M}(\mathcal{D}, u, \epsilon, \mathcal{R}) = t \in \mathcal{R}] = \frac{\exp(\epsilon u(\mathcal{D}, t)/(2\Delta u)) \cdot \mu(t)}{\sum_{r \in \mathcal{R}} \exp((\epsilon u(\mathcal{D}, r)/(2\Delta u)) \cdot \mu(r)} \tag{3.23}$$

where Δu is the sensitivity that

$$\Delta u = \max_{r \in \mathcal{R}} \max_{X, X' \in \mathbb{N}^{|\mathcal{X}|}, \|X - X'\|_1 \leq 1} |u(X, r) - u(X', r)| \tag{3.24}$$

where $\mathcal{R}$ is the range for output, and μ is a measure over $\mathcal{R}$.

In contrast to the Laplace mechanism, the exponential mechanism assigns a probability to each possible output in the group according to the utility score. By doing that, the mechanism can output a precise element using the probabilities, and perturbation is added for the selection procedure. Similarly, permute-and-flip [22] randomly chooses a value from a set of options, weighed by a utility score and the privacy parameter ϵ. For each possible output, the mechanism simulates flipping a biased coin, and the item is returned if the head is up with a probability according to an exponential function. Otherwise, it flips the coin for the next item with new probabilities assigned. Compared to the exponential mechanism, the probability of outputting an item is updated for each round, and the authors of [22] also show that the permute-and-flip mechanism never performs worse than the exponential mechanism in expectation and better in other situations.

Meanwhile, differential privacy holds sequential and parallel composition properties [23] as shown in Theorems 2 and 3. For a series of mechanisms $\mathcal{M}_{[k]} = (\mathcal{M}_1, \ldots, \mathcal{M}_k)$, each $\mathcal{M}_i$ is ϵ_i-differentially private for $i \in [k]$. The sequential composition indicates that $(\sum_i \epsilon_i)$-differential privacy is guaranteed if the series of mechanisms are applied sequentially to the input, and the parallel composition implies that $(\max_i \epsilon_i)$-differential privacy is achieved if the series of mechanisms are applied to different disjoint subsets of the input.

Theorem 2 (Sequential composition). *If mechanism $\mathcal{M}_i$ is ϵ_i-differentially private, the sequence of $\mathcal{M}_{[k]}(X)$ provides $(\sum_i \epsilon_i)$-differential privacy.*

Theorem 3 (Parallel composition). *If mechanism $\mathcal{M}_i$ is ϵ_i-differentially private, and $\mathcal{D}_i$ are disjoint subsets of the input domain $\mathcal{D}$, the sequence of $\mathcal{M}_{[k]}(X \cap \mathcal{D}_i)$ provides $(\max_i \epsilon_i)$-differential privacy.*

The privacy parameter ϵ is often called the privacy budget as it can be intuitively interpreted this way. Following the intuition, a private algorithm has a total budget of ϵ that it can spend on composed operations. Operations that have little budget will return noisier results than operations with a large budget, so by composing operations in parallel when possible and by carefully distributing the budget over the operations, one can achieve a good trade-off between privacy and utility.

3.5.2 Differentially-private decision tree learning

Many previous works have proposed algorithms for training differentially private decision trees. These algorithms address privacy leakage in regular trees by replacing the node

Method			**Features**		**Mechanism**	
Name	Year	Ref	Cat.	Num.	Splitting	Labeling
SuLQ ID3	2005	[24]	●	○	$\mathcal{M}_{\text{Gaussian}}$	$\mathcal{M}_{\text{Gaussian}}$
Private-RDT	2009	[25]	●	●	-	$\mathcal{M}_{\text{Laplace}}$
SuLQ-based ID3	2010	[26]	●	●	$\mathcal{M}_{\text{Laplace}}$	$\mathcal{M}_{\text{Laplace}}$
DiffPID3	2010	[26]	●	●	$\mathcal{M}_{EM}$	$\mathcal{M}_{\text{Laplace}}$
DiffGen	2011	[27]	●	●	$\mathcal{M}_{EM}$	$\mathcal{M}_{\text{Laplace}}$
DT-Diff	2013	[28]	●	●	$\mathcal{M}_{EM}$	$\mathcal{M}_{\text{Laplace}}$
dpRFMV/dpRFTA	2014	[29]	●	●	-	$\mathcal{M}_{\text{Laplace}}$
DPDF	2015	[30]	●	○	$\mathcal{M}_{EM}$	$\mathcal{M}_{\text{Laplace}}$
Rana et al.	2015	[31]	●	●	$\mathcal{M}_{\text{Laplace}}$	$\mathcal{M}_{\text{Laplace}}$
Smooth Random	2017	[32]	●	●	-	$\mathcal{M}_{EM}{}^*$
ADiffP	2018	[33]	●	○	$\mathcal{M}_{EM}$	$\mathcal{M}_{\text{Laplace}}$
DPGDF	2019	[34]	●	○	$\mathcal{M}_{EM}$	$\mathcal{M}_{EM}{}^*$
BDPT	2020	[35]	●	●	$\mathcal{M}_{EM}{}^*$	$\mathcal{M}_{\text{Laplace}}$
TrainSingleTree	2020	[36]	○	●	$\mathcal{M}_{EM}$	$\mathcal{M}_{\text{Laplace}}$
DiffPrivLib	2021	[37]	○	●	-	$\mathcal{M}_{PF}$
PrivaTree	*This work*		●	●	$\mathcal{M}_{PF}{}^*$	$\mathcal{M}_{PF}$

Table 3.3: Overview of methods for training differentially private decision trees, algorithms marked with * use smooth sensitivity. Most methods use the exponential mechanism $\mathcal{M}_{EM}$ for splitting and $\mathcal{M}_{\text{Laplace}}$ for labeling leaves. Methods without splitting a mechanism use random trees.

splitting and leaf labeling operations with differentially private alternatives. Fletcher and Islam [38] wrote a survey on this topic which also examines ensembles such as in private boosting [39].

Table 3.3 summarizes existing algorithms for training private decision trees. In the ‘features’ column, we indicate whether the algorithm considers categorical and numerical features. We remark that algorithms for numerical splits can also support categories using one-hot encoding, and algorithms for categories can heuristically support numerical features by applying binning. Note that unless computed using differential privacy, the resulting bins reveal information about the training data. In that case, the model only guarantees differential privacy for the leaves, which is equivalent to labelDP [40]. The mechanism columns of the table indicate the private mechanisms used for node splitting and leaf labeling. Besides the specific choice of mechanism, the way these mechanisms are sequentially applied and the way the privacy budget is distributed have a significant effect on the algorithm’s performance.

There are two main categories of algorithms for training differentially-private trees. The first category trains random decision trees, that replace split searching by splitting uniformly at random from the domain of possible feature values. A benefit of doing so is that splitting does not depend on the data and does not consume any private budget, so labeling can be performed within the complete budget. However, random splits do not necessarily produce good leaves, as having worse splits leads to leaves that contain a mix of samples from all classes. As a result, accurate labels will still cause misclassifications.

For certain datasets, the poor quality of random splits strongly affects the performance of the resulting tree. For this reason, random decision trees are almost exclusively used in ensembles. Examples of such algorithms are Private-RDT [25], dpRFMV/dpRFTA [29], Smooth Random Trees [32] and the implementation of DiffPrivLib trees [37].

The second category consists of algorithms that train a greedy tree by probabilistically choosing a split weighed by a scoring function such as the information gain or the Gini impurity. SuLQ ID3 [24] and SuLQ-based ID3 [26] do so by adding Gaussian or Laplace noise to the scores themselves, while works like DiffPID3 [26], DiffGen [27], DT-Diff [28] and TrainSingleTree [36] do so not by perturbing the scores, but using the exponential mechanism so the privacy budget does not have to be divided over so many queries. DPDF [30] further increases the utility of the queries by bounding the sensitivity of the Gini impurity, while ADiffP [33] dynamically allocates the privacy budget.

We compare against three of the latest algorithms for training private trees. BDPT [35] is a greedy tree algorithm that uses the exponential mechanism for splitting but with smooth sensitivity, allowing for a higher utility per query than previous works. DPGDF [34] is a similar algorithm that uses smooth sensitivity for creating the leaves rather than the splits. This algorithm only supports categorical features. Finally, DiffPrivLib [37] offers a recent implementation of random trees. For the leaves, it uses the permute-and-flip mechanism, which performs better in practice than the exponential mechanism [22]. This algorithm only supports numerical features.

3.5.3 Provable poisoning robustness

While it is difficult to provide strong guarantees for poisoning robustness, several works have done so in the past. Steinhardt et al. [41] remove outliers based on metrics such as distance from the centroid of a class and can formally guarantee the accuracy of convex learners under specific poisoning attacks. Rosenfeld et al. [42] consider label-flipping attacks in which only the labels can be changed by the attacker. They certify robust prediction of specific test samples for linear models by giving a bound on the minimum number of training labels that must be flipped to change the test sample's prediction. Deep Partition Aggregation (DPA) [43] trains an ensemble on disjoint subsets of the training dataset such that each sample is seen by only one ensemble member. Then, for a specific test sample, one can compute a bound on the number of train samples that should be perturbed to flip the prediction. These previous methods do not apply to single decision tree models.

Several works have guaranteed robustness using differential privacy. Ma et al. [44] prove general bounds on the attack cost for varying numbers of poisoned samples and apply this analysis to logistic regression. Also, several works have used differentially private neural networks to be more robust against poisoning attacks [45–47], sometimes even demonstrating robustness in configurations in which the theoretical guarantees do not apply. Unlike DPA, differential privacy guarantees the robustness of a global metric such as accuracy, while methods like DPA guarantee robust predictions for each test sample individually. The link between robustness and privacy is also explored in several works [48–50].

Bibliography

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks,* arXiv preprint arXiv:1312.6199 (2013).

[2] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, *Evasion attacks against machine learning at test time,* in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13* (Springer, 2013) pp. 387–402.

[3] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, *Robust decision trees against adversarial examples,* arXiv preprint arXiv:1902.10660 (2019).

[4] Y. Wang, H. Zhang, H. Chen, D. Boning, and C.-J. Hsieh, *On lp-norm robustness of ensemble decision stumps and trees,* in *International Conference on Machine Learning* (PMLR, 2020) pp. 10104–10114.

[5] A. Kantchelian, J. D. Tygar, and A. Joseph, *Evasion and hardening of tree ensemble classifiers,* in *International conference on machine learning* (PMLR, 2016) pp. 2387–2396.

[6] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards deep learning models resistant to adversarial attacks,* in *International Conference on Learning Representations* (2018).

[7] N. Justin, S. Aghaei, A. Gomez, and P. Vayanos, *Optimal robust classification trees,* in *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond* (2021).

[8] P. Mohajerin Esfahani and D. Kuhn, *Data-driven distributionally robust optimization using the wasserstein metric: Performance guarantees and tractable reformulations,* Mathematical Programming **171**, 115 (2018).

[9] S. Shafieezadeh Abadeh, P. M. Mohajerin Esfahani, and D. Kuhn, *Distributionally robust logistic regression,* Advances in neural information processing systems **28** (2015).

[10] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, *Treant: training evasion-aware decision trees,* Data Mining and Knowledge Discovery **34**, 1390 (2020).

[11] M. Andriushchenko and M. Hein, *Provably robust boosted decision stumps and trees against adversarial attacks,* arXiv preprint arXiv:1906.03526 (2019).

[12] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, *Manipulating machine learning: Poisoning attacks and countermeasures for regression learning,* in *2018 IEEE symposium on security and privacy (SP)* (IEEE, 2018) pp. 19–35.

[13] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, *Detecting backdoor attacks on deep neural networks by activation clustering,* arXiv preprint arXiv:1811.03728 (2018).

[14] E. Wong and Z. Kolter, *Provable defenses against adversarial examples via the convex outer adversarial polytope,* in *International conference on machine learning* (PMLR, 2018) pp. 5286–5295.

[15] A. Raghunathan, J. Steinhardt, and P. Liang, *Certified defenses against adversarial examples,* arXiv preprint arXiv:1801.09344 (2018).

[16] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, *Certified robustness to adversarial examples with differential privacy,* in *2019 IEEE symposium on security and privacy (SP)* (IEEE, 2019) pp. 656–672.

[17] C. Dwork, *Differential privacy,* in *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, Lecture Notes in Computer Science, Vol. 4052, edited by M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener (Springer, 2006) pp. 1–12.

[18] C. Dwork and A. Roth, *The algorithmic foundations of differential privacy,* Found. Trends Theor. Comput. Sci. **9**, 211 (2014).

[19] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, *Calibrating noise to sensitivity in private data analysis,* in *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, Lecture Notes in Computer Science, Vol. 3876, edited by S. Halevi and T. Rabin (Springer, 2006) pp. 265–284.

[20] A. Ghosh, T. Roughgarden, and M. Sundararajan, *Universally utility-maximizing privacy mechanisms,* SIAM J. Comput. **41**, 1673 (2012).

[21] F. McSherry and K. Talwar, *Mechanism design via differential privacy,* in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings* (IEEE Computer Society, 2007) pp. 94–103.

[22] R. McKenna and D. R. Sheldon, *Permute-and-Flip: A new mechanism for differentially private selection,* Advances in Neural Information Processing Systems **33**, 193 (2020).

[23] F. McSherry, *Privacy integrated queries: an extensible platform for privacy-preserving data analysis,* in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, edited by U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul (ACM, 2009) pp. 19–30.

[24] A. Blum, C. Dwork, F. McSherry, and K. Nissim, *Practical privacy: the sulq framework,* in *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA,* edited by C. Li (ACM, 2005) pp. 128–138.

[25] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, *A practical differentially private random decision tree classifier,* Trans. Data Priv. **5**, 273 (2012).

[26] A. Friedman and A. Schuster, *Data mining with differential privacy,* in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, edited by B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang (ACM, 2010) pp. 493–502.

[27] N. Mohammed, R. Chen, B. C. M. Fung, and P. S. Yu, *Differentially private data release for data mining,* in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, edited by C. Apté, J. Ghosh, and P. Smyth (ACM, 2011) pp. 493–501.

[28] T. Zhu, P. Xiong, Y. Xiang, and W. Zhou, *An effective deferentially private data releasing algorithm for decision tree,* in *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2013 / 11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13 / 12th IEEE International Conference on Ubiquitous Computing and Communications, IUCC-2013, Melbourne, Australia, July 16-18, 2013* (IEEE Computer Society, 2013) pp. 388–395.

[29] M. Bojarski, A. Choromanska, K. Choromanski, and Y. LeCun, *Differentially- and non-differentially-private random decision trees,* CoRR **abs/1410.6973** (2014), 1410.6973 .

[30] S. Fletcher and M. Z. Islam, *A differentially private decision forest,* in *Thirteenth Australasian Data Mining Conference, AusDM 2015, Sydney, Australia, August 2015*, CRPIT, Vol. 168, edited by K. Ong, Y. Zhao, M. G. Stone, and M. Z. Islam (Australian Computer Society, 2015) pp. 99–108.

[31] S. Rana, S. K. Gupta, and S. Venkatesh, *Differentially private random forest with high utility,* in *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, edited by C. C. Aggarwal, Z. Zhou, A. Tuzhilin, H. Xiong, and X. Wu (IEEE Computer Society, 2015) pp. 955–960.

[32] S. Fletcher and M. Z. Islam, *Differentially private random decision forests using smooth sensitivity,* Expert Syst. Appl. **78**, 16 (2017).

[33] N. Borhan, *Budget allocation on differentially private decision trees and random forests*, Ph.D. thesis, University of Technology Sydney, Australia (2018).

[34] B. Xin, W. Yang, S. Wang, and L. Huang, *Differentially private greedy decision forest,* in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019* (IEEE, 2019) pp. 2672–2676.

[35] Z. Guan, X. Sun, L. Shi, L. Wu, and X. Du, *A differentially private greedy decision forest classification algorithm with high utility,* Comput. Secur. **96**, 101930 (2020).

[36] Q. Li, Z. Wu, Z. Wen, and B. He, *Privacy-preserving gradient boosting decision trees,* in *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020* (AAAI Press, 2020) pp. 784–791.

[37] N. Holohan, S. Braghin, P. M. Aonghusa, and K. Levacher, *Diffprivlib: The IBM differential privacy library,* CoRR **abs/1907.02444** (2019), 1907.02444 .

[38] S. Fletcher and M. Z. Islam, *Decision tree classification with differential privacy: A survey,* ACM Comput. Surv. **52**, 83:1 (2019).

[39] V. R. Asadi, M. L. Carmosino, M. Jahanara, A. Rafiey, and B. Salamatian, *Private boosted decision trees via smooth re-weighting,* arXiv preprint arXiv:2201.12648 (2022).

[40] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, and C. Zhang, *Deep learning with label differential privacy,* in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, edited by M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan (2021) pp. 27131–27145.

[41] J. Steinhardt, P. W. W. Koh, and P. S. Liang, *Certified defenses for data poisoning attacks,* Advances in neural information processing systems **30** (2017).

[42] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter, *Certified robustness to label-flipping attacks via randomized smoothing,* in *International Conference on Machine Learning* (PMLR, 2020) pp. 8230–8241.

[43] A. Levine and S. Feizi, *Deep partition aggregation: Provable defenses against general poisoning attacks,* in *International Conference on Learning Representations* (2021).

[44] Y. Ma, X. Zhu, and J. Hsu, *Data poisoning against differentially-private learners: Attacks and defenses,* arXiv preprint arXiv:1903.09860 (2019).

[45] S. Hong, V. Chandrasekaran, Y. Kaya, T. Dumitraş, and N. Papernot, *On the effectiveness of mitigating data poisoning attacks with gradient shaping,* arXiv preprint arXiv:2002.11497 (2020).

[46] J. Geiping, L. H. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein, *Witches' brew: Industrial scale data poisoning via gradient matching,* in *International Conference on Learning Representations* (2020).

[47] E. Borgnia, J. Geiping, V. Cherepanova, L. Fowl, A. Gupta, A. Ghiasi, F. Huang, M. Goldblum, and T. Goldstein, *Dp-instahide: Provably defusing poisoning and backdoor attacks with differentially private data augmentations,* arXiv preprint arXiv:2103.02079 (2021).

[48] C. Dwork and J. Lei, *Differential privacy and robust statistics,* in *Proceedings of the forty-first annual ACM symposium on Theory of computing* (2009) pp. 371–380.

[49] K. Georgiev and S. Hopkins, *Privacy induces robustness: Information-computation gaps and sparse mean estimation,* Advances in Neural Information Processing Systems **35**, 6829 (2022).

[50] S. B. Hopkins, G. Kamath, M. Majid, and S. Narayanan, *Robustness implies privacy in statistical estimation,* in *Proceedings of the 55th Annual ACM Symposium on Theory of Computing* (2023) pp. 497–506.

4 Efficient Training of Robust Decision Trees

Current state-of-the-art algorithms for training robust decision trees have high runtime costs and require hours to run. We present GROOT, an efficient algorithm for training robust decision trees and random forests that runs in a matter of seconds to minutes. Where before the worst-case Gini impurity was computed iteratively, we find that we can solve this function analytically to improve time complexity from $\mathcal{O}(n)$ to $\mathcal{O}(1)$ in terms of n samples. Our results on both single trees and ensembles on 14 structured datasets as well as on MNIST and Fashion-MNIST demonstrate that GROOT runs several orders of magnitude faster than the state-of-the-art works and also shows better performance in terms of adversarial accuracy on structured data.

This chapter is based on *D. Vos and S. Verwer. Efficient Training of Robust Decision Trees Against Adversarial Examples, ICML 2021* [1].

4.1 Introduction

Recently it has been shown that neural networks [2, 3] and similarly linear models, decision trees and support vector machines [4] are vulnerable to adversarial examples: perturbed samples that trick the model into misclassifying them. Much research has gone into training robust neural networks [5–8]. These models perform well on unstructured data such as images and audio, but decision tree ensembles often outperform them on structured data. Additionally, when using a single decision tree, the models are easily interpreted by humans. Recently the first methods have been proposed to train decision trees and their ensembles robustly [9–12] but the state-of-the-art methods are expensive to run.

In this work we propose GROOT, an efficient algorithm for training robust decision trees. Like Chen et al. [10], we closely mimic the greedy recursive splitting strategy that traditional decision trees use and we score splits with the adversarial Gini impurity. We prove that the adversarial Gini impurity is concave with respect to the number of modified data points and use its analytical solution to compute the function in constant time. Our results show that GROOT trains trees 3 to 6 orders of magnitude faster than the state-of-the-art method TREANT [11] and GROOT trains random forests 100-1000 times faster than provably robust boosting [12]. Leveraging this speedup we can fit robust random forests using the adversarial Gini impurity and we do not have to rely on a heuristic such as Chen et al.

Moreover, GROOT scores competitively on adversarial accuracy which we evaluate on 14 structured datasets as well as on MNIST and Fashion-MNIST. On the structured data, both GROOT trees and GROOT forests outperform the state-of-the-art robust tree and forest methods. GROOT trees obtain a small performance improvement over TREANT. GROOT forests outperform provably robust boosting. Interestingly, GROOT trees and forests obtain top and similar ranks. Showing that in contrast to regular accuracy, there is not much difference between the adversarial accuracy obtained by robust decision trees and forests. On MNIST and Fashion-MNIST, provably robust boosting outperforms GROOT forests on robustness by respectively 0.8% and 5.5% but takes 122 times and 162 times longer to train. We implement and publish GROOT's source code in a Scikit-learn [13] compatible classifier. We take inspiration from TREANT and allow users to easily configure the perturbation range for each separate feature. Our main contributions are:

- An efficient score function that allows us to fit trees orders of magnitude faster than the state of the art.
- An algorithm that achieves competitive performance to the state of the art in the adversarial setting.
- A flexible implementation that allows users to specify attacks in terms of axis aligned perturbations.

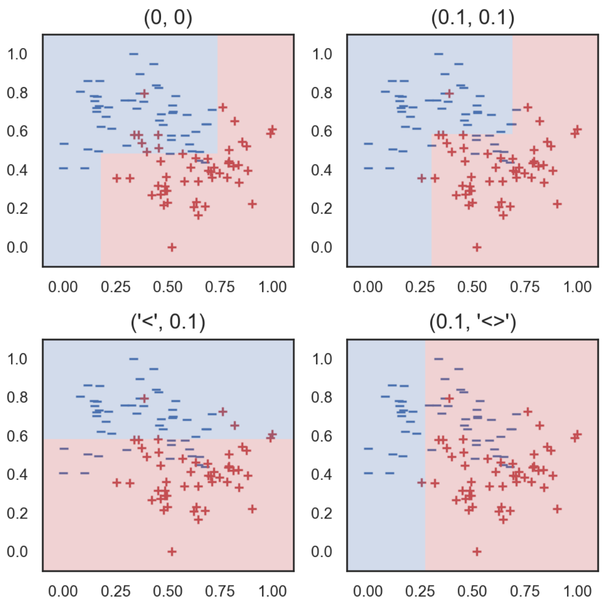


Figure 4.1: Decision regions of GROOT trees attacked by different threat models (indicated above each image). The threat model greatly influences the learned trees, e.g. robust decision trees against L^∞ perturbations (top right) are different from trees robust against other attackers (bottom).

4.2 Specifying Threat Models

In our work, we assume the existence of an attacker that knows the model and perturbs samples according to a user-specified threat model. Assuming that the attacker knows the model means that we are also protected from attacks that assume only 'black-box' access to the model. To support a wide range of attack types we take inspiration from TREANT [11] and let the user define the perturbation limits for each individual feature. The specification is as follows:

- ""or None: This feature cannot be perturbed.
- $>$ or $<$: This feature can be increased / decreased.
- $>$: This feature can only be perturbed to a higher value.
- $<$: This feature can only be perturbed to a lower value.
- $<>$: This feature can be perturbed to any value.
- ϵ: The feature can be perturbed by a distance of ϵ.
- (ϵ_l, ϵ_r): The feature can be perturbed ϵ_l left or ϵ_r right.

We visualize GROOT trees with a variety of threat models in Figure 4.1. It is worth noting that all these cases can be translated to the tuple notation, e.g. we can encode $>$ as $(0, \infty)$ or a number ϵ as (ϵ, ϵ). For conciseness we only use the tuple notation in the algorithms in section 4.4. When we set the threat model to ϵ for each feature, it behaves identically to an L^∞ norm. We also allow the user to choose whether one or both of the classes can be perturbed.

In the rest of the chapter we use an L^∞ threat model where both classes move and where features are scaled to the range $[0,1]$ since this allows us to compare to existing work. To foster further research we implemented GROOT according to the Scikit-learn API and published the code on GitHub[1].

4.3 Adversarial Gini Impurity

Similar to robust decision trees [10] we use the worst-case Gini impurity to score threshold values. We show that we can efficiently compute this function by leveraging its concavity.

4.3.1 Adversarial Gini Impurity for Two Moving Classes

We typically fit decision trees with a splitting criterion such as the Gini impurity. To determine the quality of a split we then take the weighted average of the scores on both sides. We can define the Gini impurity for two classes as:

$$G(n_0,n_1) = 1 - \left(\tfrac{n_0}{n_0+n_1}\right)^2 - \left(\tfrac{n_1}{n_0+n_1}\right)^2 \tag{4.1}$$

Where n_0 and n_1 are the number of samples of label 0 and 1 respectively. Then we combine this into a score function by taking the weighted average with respect to number of samples on each side of the split (other works use the Gini gain which behaves identically):

$$S(l_0,l_1,r_0,r_1) = \frac{(l_0+l_1)\cdot G(l_0,l_1) + (r_0+r_1)\cdot G(r_0,r_1)}{l_0+l_1+r_0+r_1} \tag{4.2}$$

Where l_0 and l_1 are the number of samples on the left side of the split of label 0 and 1 respectively. Similarly r_0 and r_1 represent samples on the right. Normally one searches for a split that minimizes this score function. Instead, we keep track of a set I that contains all samples close enough to the split to cross it under adversarial influence. We minimize the score function after the attacker maximizes it by perturbing the samples in I.

Where one normally minimizes the Gini impurity, we assume an attacker that aims to maximize $S(l_0,l_1,r_0,r_1)$ by moving samples from I to different sides of the split. We visualize this maximization problem in Figure 4.2. Here, i_1 is the number of points with label 1 that are close enough to the split that the adversary can move them to either side. Mathematically we are looking for the integer $m_1 \in [0,i_1]$ such that m_1 points of I_1 move to the left side of the split and $i_1 - m_1$ to the right. Similarly we have an i_0 and m_0 for the class 0 samples. The score function under attacker influence is:

$$S_{\text{robust}}(l_0,l_1,r_0,r_1,i_0,i_1) = \max_{m_1\in[0,i_1],m_0\in[0,i_0]} S(l_0+m_0,l_1+m_1,r_0+i_0-m_0,r_1+i_1-m_1) \tag{4.3}$$

We can then write the m_1' and m_0' that maximize it as:

$$m_1',m_0' = \operatorname*{arg\,max}_{m_1\in[0,i_1],m_0\in[0,i_0]} \left(\frac{(l_0+m_0)(l_1+m_1)}{l_0+l_1+m_1+m_0} + \frac{(r_0+i_0-m_0)(r_1+i_1-m_1)}{r_0+r_1+i_0+i_1-m_1-m_0} \right) \tag{4.4}$$

The algorithm by Chen et al. [10] optimizes a similar function by iterating through the I samples to perform gradient ascent. However, since the function is concave with respect

[1]`https://github.com/tudelft-cda-lab/GROOT`

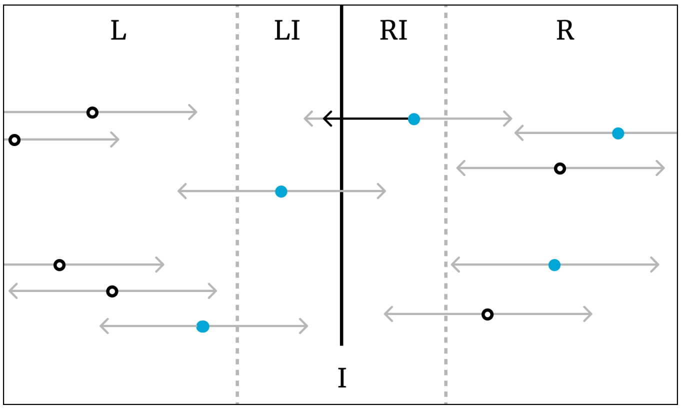


Figure 4.2: Example of the adversarial Gini impurity where samples can move in the range of the arrows. We want to move a number of samples from I over the threshold (line, center) to maximize the weighted average of Gini impurities. In this example we can move the single blue (filled) sample from RI into LI to maximize it.

to x and y, we can do this faster by maximizing the function analytically and rounding to a near integer solution. The maxima form the following line (proofs in the appendix[2]):

$$m_0' = \frac{l_1(r_0+i_0) - l_0(r_1+i_1)}{l_1+r_1+i_1} + \frac{(l_0+r_0+i_0)m_1'}{l_1+r_1+i_1} \tag{4.5}$$

Using gradient ascent and given enough movable samples one would end up on the optimal line, but not necessarily on the closest point on the line to the starting values for m_1 and m_0. We argue that the closest point is intuitively a better solution than a random point on the line, because it represents the least number of samples to move. Therefore in GROOT we find the closest point on the solution line to the starting values of m_1, m_0 ($|LI_1|, |LI_0|$) then round to the nearest integers.

Computing the point and rounding it takes $\mathcal{O}(1)$ time. Therefore we have an efficient method (constant time) to compute S_{robust} (Equation 4.2).

4.4 GROOT

We introduce GROOT (Growing RObust Trees), an algorithm that trains decision trees that are robust against adversarial examples generated from a user-specified threat model. The algorithm stays close to regular decision tree learning algorithms but searches through more candidate splits with a robust score function and propagates samples according to an attacker. Like regular decision tree learning algorithms, GROOT runs in $\mathcal{O}(n \log n)$ time in terms of n samples. Similar to these algorithms, GROOT greedily makes splits according to a heuristic. This strategy performs well in practice but has no provable bound [14].

4.4.1 Scoring Candidate Splits

Similar to regular decision tree learning algorithms we can search over all possible splits and compute a score function to find the best split. In Algorithm 4 we iterate over each

[2]`https://arxiv.org/abs/2012.10438`

Algorithm 4 Find Best Robust Split on Numerical feature

```
Input: feature values $X$, perturbation limits $(\epsilon_l, \epsilon_r)$
$X_1$ refers to the samples with label 1
 1: $S \leftarrow X \cup \{o - \epsilon_l | o \in X\} \cup \{o + \epsilon_r | o \in X\}$
 2: for $s \in S$ do
 3:     $R \leftarrow \{o | o \in X \land o > s + \epsilon_r\}$
 4:     $RI \leftarrow \{o | o \in X \land s < o \leq s + \epsilon_r\}$
 5:     $LI \leftarrow \{o | o \in X \land s - \epsilon_l < o \leq s\}$
 6:     $L \leftarrow \{o | o \in X \land o \leq s - \epsilon_l\}$
 7:     $(m_{1s}, m_{0s}) \leftarrow$ number of samples from $I$ to move left    ▷ See Equation 4.5
 8:     $m_{1s} \leftarrow \text{round}(m_{1s}), \quad m_{0s} \leftarrow \text{round}(m_{0s})$
 9:     $g_s \leftarrow S(|L_0| + m_{0s},\ |L_1| + m_{1s},\ |R_0| + |I_0| - m_{0s},\ |R_1| + |I_1| - m_{1s})$    ▷ See Equation 4.2
10: end for
11: $s' \leftarrow \arg\min_s\ g_s$
12: split with threshold $s'$
Output: $(s', g_{s'}, m_{1s'}, y_{s'})$
```

Algorithm 5 Fit Robust Tree on Numerical Data

```
Input: sample set $X$, perturbation limits $(\epsilon_l, \epsilon_r)$
$X_1$ refers to the samples with label 1
 1: if stopping criterion (e.g. maximum depth) then
 2:     create Leaf($|X_0|, |X_1|$)
 3: else
 4:     for $f \leftarrow 1 \dots F$ do
 5:         $s_f, g_f, m_{1f}, m_{0f} \leftarrow$ BestRobustSplit($X^f, \epsilon_l, \epsilon_r$)
 6:     end for
 7:     $f' \leftarrow \arg\min_f\ g_f$
 8:     determine $R$, $RI$, $LI$, $L$ for split $f'$ as in Alg. 4
 9:     move $m_{1f'}$ random samples from $I_1$ to $LI_1$, move remainder in $I_1$ to $RI_1$
10:     move $m_{0f'}$ random samples from $I_0$ to $LI_0$, move remainder in $I_0$ to $RI_0$
11:     node$_l \leftarrow$ FitRobustTree($L \cup LI$)
12:     node$_r \leftarrow$ FitRobustTree($R \cup RI$)
13:     create DecisionNode($s_{f'}$, node$_l$, node$_r$)
14: end if
```

sample in sorted order to identify candidate splits. We evaluate each candidate split with the adversarial Gini impurity from Section 4.3 to find the split that is accurate against an adversary. Below, we describe our algorithm for numerical features; for categorical features, we refer to the Appendix[3]. The time complexity in terms of n samples for both cases is bounded by $\mathcal{O}(n \log n)$ per feature.

In regular decision tree learning algorithms we score candidate splits at each position in which a sample moves from the right to the left side. However, when an adversary can

[3]`https://arxiv.org/abs/2012.10438`

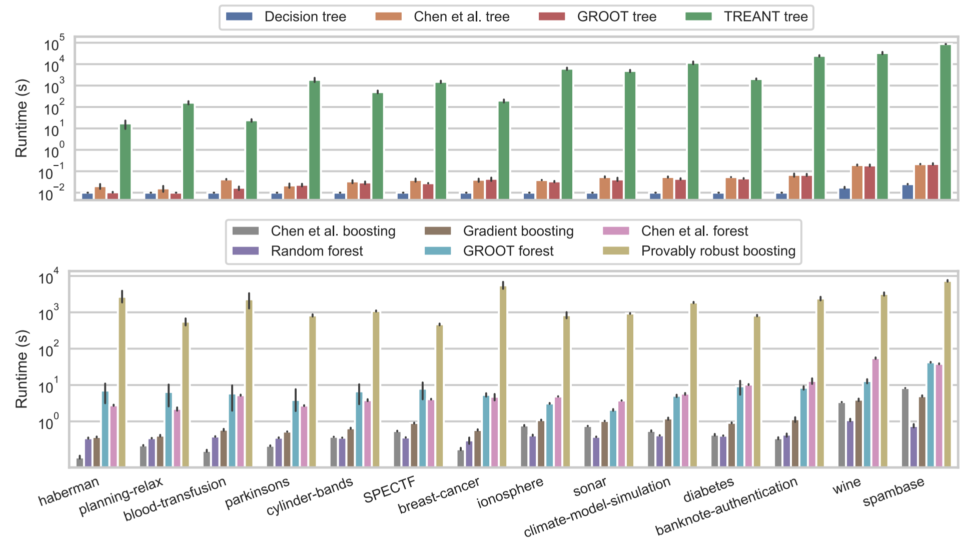


Figure 4.3: Runtimes of decision trees and ensembles in seconds on a logaritmic scale. Decision trees, random forests and gradient boosting enjoy Scikit-learn's optimized implementation. GROOT and Chen et al. consistently run 100-1000 times faster than TREANT and provably robust boosting. TREANT's spambase runs were terminated after 24 hours.

perturb samples there are more possible splits that affect the sample counts on each side. Therefore we consider also candidate splits where a movable sample becomes in or out of range of I. Take for example a sample at position 0.4 that can be perturbed in a radius of 0.1, we score a split at 0.3, 0.4 and 0.5. At the start of Algorithm 4 we sort all candidate splits. We can then compute the sample counts and evaluate each split in $\mathcal{O}(1)$ time, as explained in Section 4.3. Recall that we consider at maximum $3n$ splits, where n is number of samples. Therefore the time complexity of evaluating splits is $\mathcal{O}(n)$ per feature and this means the fitting run time is dominated by the sorts of complexity $\mathcal{O}(n \log n)$.

4.4.2 Propagating Samples

When fitting regular decision trees one can simply move all samples lower than a threshold left and higher to the right. In our robust trees we account for samples that the adversary moves by modifying this propagation which we define in Algorithm 5. We do not only keep track of left (L) and right (R) samples, but also store an 'intersection' set $I = LI \cup RI$ that contains samples that can move to both sides. Here LI are the samples from I that were originally on the left side and RI were originally on the right side.

In section 4.3 we showed that m_1' and m_0' are the optimal values for the adversarial Gini impurity. From I_1 we move samples over the split to place m_1' samples on the left and $I_1 - m_1'$ on the right. If there were, before moving, fewer than m_1' samples on the left we move samples from the right. If there were more samples on the left we move them to the right. We do this to keep as many samples as possible on the original side of the split. We repeat the same procedure for m_0 and I_0.

The actual samples that move are randomly selected from the intersection I this makes our algorithm non-deterministic and therefore with different randomization seeds the algorithm can fit different trees. Our intuition behind random selection is that it prevents influence on the data distribution in splits further down the tree. If we were to move e.g. the closest samples to the threshold over the split, these samples might correlate with other features and cause any side of the split to become biased to specific values of that feature. The strategy differs between methods, e.g. TREANT chooses to move a sample to the side where it currently incurs the greatest loss. In future works, one could fit trees by optimizing all splits at once instead of a greedy method. In that case it is not needed to implement a sample propagation strategy but algorithms for optimal decision trees come at the cost of runtime.

4.4.3 GROOT Random Forests

To enable its use in ensembles we also implement a random forest of GROOT decision trees. In random forests it is important that the individual models have low covariance which we achieve using the same techniques as regular random forests [15]. Specifically, we train each decision tree on a bootstrap sample of the original training set and limit each decision node to scanning a random selection of $\sqrt{f}$ features (given f the total number of features). We do not limit the size of the decision trees. Similarly to Scikit-learn's implementation of random forests, the ensemble makes predictions by averaging and then rounding all individual tree predictions.

4.5 Results

We present results on 14 structured datasets as well as MNIST [16] and Fashion-MNIST [17]. We compare GROOT against regular tree based models, the methods by Chen et al., TREANT and provably robust boosting. For the regular models we use scikit-learn's [13] implementation as it is widely used for research in the field. The used hyperparameters are summarized in Table 4.4. All datasets can be retrieved from OpenML[4], their specific versions, size and corresponding ϵ values can be found in Table 4.1. We removed any data row with missing values as it is unclear how to measure robustness against these samples.

4.5.1 Training Runtime

To compare the efficiency of the algorithms, we plot the run times of each run in Figure 4.3, the results shown are averaged over 5 data folds. All experiments ran on a Linux machine with 16 Intel Xeon CPU cores and 72GB of RAM total. Each algorithm instance ran on a single core and therefore did not use any parallel optimizations.

Regarding the single decision tree models, regular decision trees enjoy the optimized code by Scikit-learn which is clearly the fastest. Comparing TREANT and GROOT, we see that our algorithm runs three to six orders of magnitude faster. TREANT exhaustively searches for attacks using an exponential search and uses a sequential quadratic programming solver to optimize the loss function which likely contributes to the higher run time. The heuristic by Chen et al. has a similar runtime as GROOT as it only uses a different splitting criterion, which we implemented in the code of GROOT.

[4]`https://www.openml.org/`

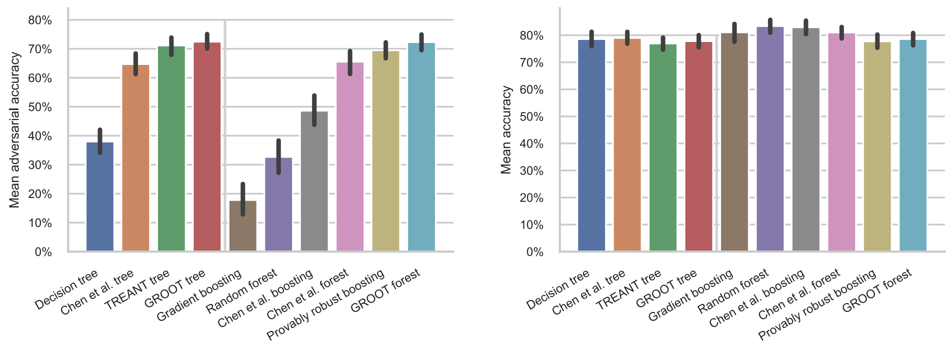


Figure 4.4: Average adversarial accuracy (left) and accuracy scores (right) over 13 structured datasets. TREANT and GROOT fit the most robust decision trees while provably robust boosting and GROOT random forests fit the most robust ensembles. More robust models tend to score up to 5% worse on regular accuracy.

In the ensemble model results we again see very fast results from the optimized implementations by Scikit-learn (Random forest and Gradient boosting) and Chen et al. boosting which is built on XGBoost [18]. Still, GROOT and the Chen et al. forest run 2 to 3 orders of magnitude faster than provably robust boosting.

4.5.2 Predictive Performance on Structured Data

To determine the quality of the models produced by each algorithm we measure adversarial accuracy using the exact MILP attack [9] which we modified to a feasibility problem as done in [12] to improve run time. The adversarial accuracy is the accuracy after samples have been optimally perturbed within an L^∞ ball of radius ϵ.

We encoded the above threat models in TREANT's attack rules using precondition $[-\infty, \infty]$ and postcondition ϵ or $-\epsilon$. Each rule has cost 1 and the attacker has a budget equal to the depth of the trees. In the original TREANT implementation, it allows attack rules to be applied to the same feature multiple times (in each decision node), resulting in attacks larger than ϵ. To exactly match the threat model specifications, we modify TREANT to only use attack rules once per feature. Preliminary testing without this modification gave poor results. Given these modifications the attack rules exactly encode the L^∞ radius attack model.

Another popular method for measuring robustness (e.g. in [9, 10]) is to compute the average perturbation distance required to cause a misclassification. We choose against this metric as it assumes features can perturb arbitrarily and with equal cost.

We train each model on each dataset with 5 fold stratified cross validation. All single decision trees were trained up to a depth of 4 to maintain interpretability. We implemented GROOT and the heuristic by Chen et al. in Python but used TREANT's existing implementation with before-mentioned minor modifications. Scikit-learn uses the Gini impurity and TREANT optimizes the sum of squared errors. All models required at least 10 samples to make a split and 5 samples to create a leaf. We allowed all models to split multiple times on the same feature. All ensembles were limited to training 100 trees. We report the average adversarial accuracy and regular accuracy over 13 of the 14 structured

Dataset	Samples	Features	ϵ
banknote-authentication (1)	1372	4	0.1
blood-transfusion (1)	748	4	0.1
breast-cancer (1)	683	9	0.3
climate-model-simulation (4)	540	18	0.1
cylinder-bands (2)	277	37	0.1
diabetes (1)	768	8	0.05
haberman (1)	306	3	0.1
ionosphere (1)	351	34	0.2
parkinsons (1)	195	22	0.1
planning-relax (1)	182	12	0.1
sonar (1)	208	60	0.1
spambase (1)	4601	57	0.05
SPECTF (2)	267	44	0.1
wine (1)	6497	11	0.05

Table 4.1: Structured datasets used by OpenML (version) name. Whenever possible, ϵ is taken from earlier work.

datasets in Figure 4.4. We present the number of wins and average ranks over the datasets in Table 4.2. In both, we left out the results on spambase as TREANT did not finish fitting after multiple days of running.

While all models considered scored well on accuracy, the scores in the adversarial setting differ. Regarding single trees, TREANT and GROOT perform similarly on adversarial accuracy and both significantly improve on regular decision trees by approximately 33%. GROOT obtains just over 1% more adversarial accuracy on average than TREANT, only one more win, but does obtain a higher mean overall rank. The heuristic by Chen et al. score approximately 7% worse than TREANT and GROOT. The individual results for each dataset are given in the appendix[5].

The ensemble results show that the GROOT random forest and provably robust boosting clearly achieve the best adversarial accuracy scores by about 5% difference over Chen et al. forest. Moreover, GROOT performs about 2.5% better than provably robust forest on average and obtains the best mean rank across all methods. The results from Chen et al. boosting were significantly lower than the scores from the random forest. We expect that the boosting model is more sensitive to the specific hyperparameters and that one could improve the scores using a hyperparameter search.

Interestingly, there is no clear difference between the best ensemble models and single decision tree models with regards to adversarial accuracy. This is in contradiction with the regular accuracy scores of decision trees and random forests. In those scores we see a clear 5% difference.

[5]`https://arxiv.org/abs/2012.10438`

Model	Nr. Wins	Mean rank
Chen et al. boosting	0	7.5
Chen et al. forest	3	4.3
Chen et al. tree	0	5.1
Decision tree	0	7.9
GROOT forest	**9**	**1.5**
GROOT tree	7	1.8
Gradient boosting	0	9.8
Provably robust boosting	2	3.7
Random forest	0	8.2
TREANT tree	6	2.7

Table 4.2: Summary of relative adversarial accuracy scores on 13 structured datasets.

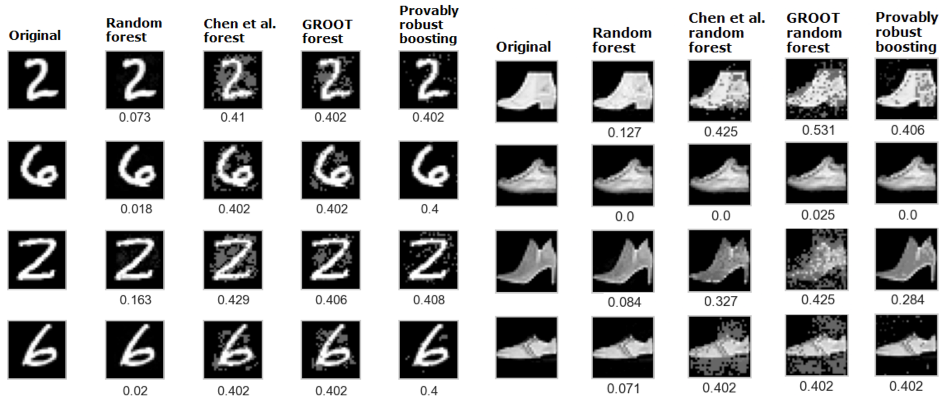
MNIST 2 vs 6			
Model	Acc.	Adv. acc.	Time
Random forest	99.7%	0.0%	3.9 sec.
Chen et al. forest	98.9%	89.1%	2.1 min.
GROOT forest	99.4%	91.9%	2.5 min.
Provably robust boosting	99.2%	92.7%	5.1 hr.
Fashion-MNIST sandals vs sneakers			
Random forest	95.8%	0.0%	5.9 sec.
Chen et al. forest	90.0%	52.6%	4.3 min.
GROOT forest	89.0%	70.4%	5.0 min.
Provably robust boosting	88.9%	75.9%	13.5 hr.

Table 4.3: Comparison of tree ensembles on image data. Provably robust boosting achieves the best adversarial accuracy but takes in the order of hours to run while GROOT runs in minutes and significantly improves on other fast models.

4.5.3 Predictive Performance on Images

To compare predictive performance on image data we present results on MNIST and Fashion-MNIST. GROOT is limited to binary classification problems so we modify the datasets to MNIST 2 vs 6 and Fashion-MNIST sandals vs sneakers, similar to what previous works have done [9, 10, 12]. The prediction scores and runtimes are given in Table 4.3. On these datasets we ran provably robust boosting with parallelization enabled. All models ran on a system with 8GB RAM and 4 Intel i7-4710MQ CPU cores (8 logical cores), the hyperparameters were the same as in the previous experiment. The datasets were randomly split in a 70%-30% stratified train-test split and were evaluated against an L^∞ radius of 0.4.

On both MNIST and Fashion-MNIST provably robust boosting achieved the best scores on adversarial accuracy. GROOT significantly improved on the adversarial accuracy scores of the other two models and scores close (0.8% difference) to the adversarial accuracy of



(a) MNIST 2 vs 6 (b) Fashion-MNIST sandals vs sneakers

Figure 4.5: Minimal adversarial examples, L^{∞} distances from the original are given below each example, larger is better. Some images already get misclassified without modification.

Parameter	Decision tree	Chen et al. tree	GROOT tree	TREANT tree	Random forest	Gradient boosting	Chen et al. boosting	Chen et al. forest	GROOT forest	Provably robust boosting
max_depth	4	4	4	4	None	8	8	None	None	8
min_samples_split	10	10	10	10	10	10	-	10	10	10
min_samples_leaf	5	5	5	-	5	5	-	5	5	5
n_estimators	-	-	-	-	100	100	100	100	100	100
η	-	-	-	-	-	-	0.2	-	-	0.2
γ	-	-	-	-	-	-	1.0	-	-	-
min_child_weight	-	-	-	-	-	-	1	-	-	-
affine	-	-	-	False	-	-	-	-	-	-

Table 4.4: Hyperparameters of all models used in our experiments. Parameters that were not applicable were left blank. Except for n_estimators, the values were copied from their original works wherever possible.

provably robust boosting on MNIST. GROOT achieves these scores while running more than 100 times faster.

Previous works [19] have described the accuracy-robustness trade-off and we find it here too. Particularly on the Fashion-MNIST dataset the most robust model sacrifices 6.9% regular accuracy where on the MNIST dataset this difference is 0.5%.

Figure 4.5 shows the minimal L^{∞} norm perturbations required to change the prediction of each model. We generate the adversarial examples using a MILP formulation for attack tree ensembles [9]. The random forest models need visibly more perturbed feature values than the provably robust boosting model so by optimizing the L^{∞} norm they also increase robustness in the L^{0} norm. We expect this is due to the random forest approach of training each decision node on a limited selection of features which causes more variety in the selected features.

4.6 Discussion and Conclusions

We present GROOT, an algorithm for learning robust decision trees. It uses an analytical solution for computing the adversarial Gini impurity and by doing so runs two to six orders of magnitude faster than the state-of-the-art approaches. Our results show that GROOT trees score competitively with TREANT and so do GROOT random forests with provably robust boosting. While the single GROOT trees are the same size as those of TREANT, there is a noticeable difference between the size of random forest and boosting models. In the case of random forests we do not limit the size of the trees, where gradient boosting trees were trained to a maximum depth of 8 (this is intended and does not reduce model performance). This means that while our method trains quickly, the models suffer from relatively longer inference and robustness verification times. In further research, the random forests may be post-processed to reduce the size.

For the sake of comparison, we experimented on the same public datasets that similar works on robustness used, but these datasets were originally not intended for research into adversarial attacks. In the near future, we will apply the methods we discussed to problems where adversarial modifications are an important concern such as fraud, malware and intrusion detection. Chen et al. [20] have successfully applied robust decision trees for such security applications. Our fast splitting criterion can also be used to speed up their algorithm.

While greedy algorithms that choose locally optimal splits are popular for fitting decision trees, they can theoretically perform arbitrarily poorly. There have been many successful efforts in training optimal decision trees [21–24] and their results show that the greedy algorithms come close to optimal performance. An optimal algorithm for robust decision trees will determine whether the greedy approaches for robust trees perform as well as their regular counterparts.

We conclude that:

- By solving the adversarial Gini impurity analytically we can now fit robust trees with the same time complexity as regular trees: $\mathcal{O}(n \log n)$, for n samples.
- This algorithm, GROOT, runs orders of magnitude faster than the state-of-the-art works and as efficiently as an existing heuristic.
- GROOT consistently achieves scores competitive with the state-of-the-art work in terms of robustness.

Bibliography

[1] D. Vos and S. Verwer, *Efficient training of robust decision trees against adversarial examples,* in *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 139, edited by M. Meila and T. Zhang (PMLR, 2021) pp. 10586–10595.

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. B. Estrach, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks,* in *2nd International Conference on Learning Representations, ICLR 2014* (2014).

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples,* arXiv preprint arXiv:1412.6572 (2014).

[4] N. Papernot, P. McDaniel, and I. Goodfellow, *Transferability in machine learning: from phenomena to black-box attacks using adversarial samples,* arXiv preprint arXiv:1605.07277 (2016).

[5] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, *Distillation as a defense to adversarial perturbations against deep neural networks,* in *2016 IEEE Symposium on Security and Privacy (SP)* (IEEE, 2016) pp. 582–597.

[6] D. Meng and H. Chen, *Magnet: a two-pronged defense against adversarial examples,* in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017) pp. 135–147.

[7] P. Samangouei, M. Kabkab, and R. Chellappa, *Defense-GAN: Protecting classifiers against adversarial attacks using generative models,* in *International Conference on Learning Representations* (2018).

[8] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry, *Adversarial examples are not bugs, they are features,* in *Advances in Neural Information Processing Systems* (2019) pp. 125–136.

[9] A. Kantchelian, J. D. Tygar, and A. Joseph, *Evasion and hardening of tree ensemble classifiers,* in *ICML* (2016) pp. 2387–2396.

[10] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, *Robust decision trees against adversarial examples,* in *ICML* (2019) pp. 1122–1131.

[11] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, *Treant: Training evasion-aware decision trees,* Data Mining and Knowledge Discovery , 1 (2020).

[12] M. Andriushchenko and M. Hein, *Provably robust boosted decision stumps and trees against adversarial attacks,* arXiv preprint arXiv:1906.03526 (2019).

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python,* Journal of Machine Learning Research **12**, 2825 (2011).

[14] M. Kearns, *Boosting theory towards practice: Recent developments in decision tree induction and the weak learning framework,* in *Proceedings of the National Conference on Artificial Intelligence* (1996) pp. 1337–1339.

[15] L. Breiman, *Random forests,* Machine learning **45**, 5 (2001).

[16] Y. LeCun, C. Cortes, and C. Burges, *Mnist handwritten digit database,* ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010).

[17] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,* arXiv preprint arXiv:1708.07747 (2017).

[18] T. Chen and C. Guestrin, *Xgboost: A scalable tree boosting system,* in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[19] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, *Robustness may be at odds with accuracy,* arXiv preprint arXiv:1805.12152 (2018).

[20] Y. Chen, S. Wang, W. Jiang, A. Cidon, and S. Jana, *Cost-aware robust tree ensembles for security applications,* in *30th USENIX Security Symposium (USENIX Security 21)* (2021).

[21] D. Bertsimas and J. Dunn, *Optimal classification trees,* Machine Learning **106**, 1039 (2017).

[22] J. Rhuggenaath, Y. Zhang, A. Akcay, U. Kaymak, and S. Verwer, *Learning fuzzy decision trees using integer programming,* in *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (IEEE, 2018) pp. 1–8.

[23] S. Verwer and Y. Zhang, *Learning optimal classification trees using a binary linear program formulation,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33 (2019) pp. 1625–1632.

[24] G. Aglin, S. Nijssen, and P. Schaus, *Learning optimal decision trees using caching branch-and-bound search,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34 (2020) pp. 3146–3153.

5 Robust Optimal Classification Trees

Decision trees are a popular choice of explainable model, but just like neural networks, they suffer from adversarial examples. Existing algorithms for fitting decision trees robustly against adversarial examples are greedy heuristics and lack approximation guarantees. In this paper we propose ROCT, a collection of methods to train decision trees that are optimally robust against user-specified attack models. We show that the min-max optimization problem that arises in adversarial learning can be solved using a single minimization formulation for decision trees with 0-1 loss. We propose such formulations in Mixed-Integer Linear Programming and Maximum Satisfiability, which widely available solvers can optimize. We also present a method that determines the upper bound on adversarial accuracy for any model using bipartite matching. Our experimental results demonstrate that the existing heuristics achieve close to optimal scores while ROCT achieves state-of-the-art scores.

This chapter is based on *D. Vos and S. Verwer. Robust Optimal Classification Trees Against Adversarial Examples, AAAI 2022* [1].

5.1 Introduction

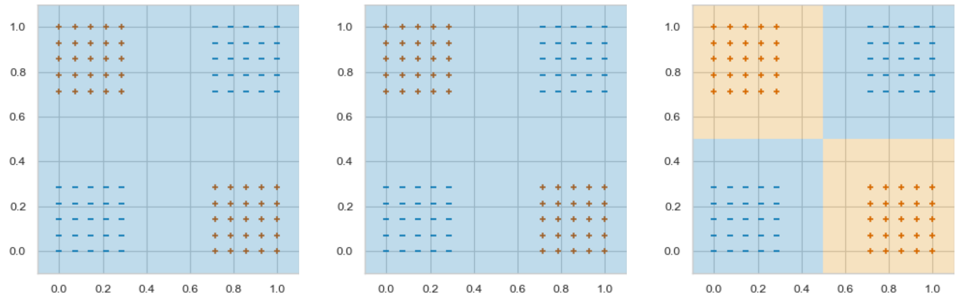
While breakthroughs in machine learning research have enabled training of powerful predictive models, most models are still vulnerable to adversarial examples, samples with tiny perturbations that cause them to be misclassified. Since the discovery of adversarial examples in neural networks [2] much work has gone into training models that are robust to these attacks and recently, the first efforts were made to train robust decision trees against adversarial examples [3–5]. However, the current methods are greedy and offer no performance guarantees. They can fail on arbitrary datasets and give results no better than random guessing (Figure 5.1).

In decision tree learning, there has been an increased interest in optimal learning algorithms [6]. Although the problem of learning decision trees is NP-complete [7], these methods can produce optimally accurate decision trees for many (typically small) datasets. Most methods translate the problem to well-known frameworks such as Mixed-Integer Linear Programming [8, 9], Boolean Satisfiability [10, 11], and Constraint Programming [12].

In this work, we combine these lines of research and propose Robust Optimal Classification Trees (ROCT), a method to train decision trees that are optimally robust against user-specified adversarial attack models. This model is robust in the sense that it predicts the correct ground-truth label in a box of specified size surrounding each sample, this optimizes robustness against corrupted instances [13]. Like existing robust decision tree learning algorithms [4, 5], ROCT allows users to specify a box-shaped attack model that encodes an attacker's capability to modify feature values with the aim of maximizing loss. Existing robust decision tree learning methods use a greedy node splitting approach. Other robust learning algorithms such as adversarial training [14] solve the inner maximization (adversarial attacks) and the outer minimization problems (minimize expected loss) separately. In this work we prove that this separation is not needed in the case of decision trees. We provide a formulation that solves the problem of fitting robust decision trees exactly in a single minimization step for trees up to a given depth.

ROCT[1] uses a novel translation of the problem of fitting robust decision trees into

[1]https://github.com/tudelft-cda-lab/ROCT



(a) GROOT (b) TREANT (c) ROCT (ours)

Figure 5.1: Existing methods (a)(b) greedily optimize one split at a time and cannot find a good tree to fit the XOR-shaped data. ROCT optimizes the entire tree at once and finds the optimal tree that exactly fits the dataset.

Mixed-Integer Linear Programming (MILP) or Maximum Satisfiability (MaxSAT) formulations. We also propose a new upper-bound calculation for the adversarial accuracy of any machine learning model based on bipartite matching, which can be used to choose appropriate attack models for experimentation. Our results show that ROCT trees optimized with a warm-started MILP solver achieve state-of-the-art adversarial accuracy scores compared to existing methods on 8 datasets. Moreover, given sufficient solver time, ROCT provably finds an optimally robust decision tree. In our experiments, ROCT was able to fit and prove optimality of depth 2 decision trees on six datasets. Where there are no known approximation bounds on the performance of existing heuristic methods for fitting robust decision trees, our results demonstrate that they are empirically close to optimal.

5.2 ROCT: Robust Optimal Classification Trees

When training robust classifiers we find ourselves in a competition with the adversary. Madry et al. [14] present the robust learning problem as the following min-max optimization problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D}\left(\max_{\delta\in S} L(\theta, x+\delta, y)\right) \tag{5.1}$$

Like in traditional machine learning, the goal is to find model parameters θ that minimize the expected loss $L(\theta, x, y)$ over feature x and class y variables from distribution D (outer minimization). This minimization takes into account that an attacker aims to maximize this loss by changing samples (x, y) from D with perturbation $\delta \in S$ (inner maximization), where S is a predefined set of allowed perturbations. Intuitively, the min-max nature of training robust models makes it a much more challenging optimization problem than regular learning. For example in adversarial training [14] one approximately optimizes this function by incorporating expensive adversarial attacks into the training procedure. In this work, we demonstrate that this intuition is wrong when learning decision trees.

Let $\mathcal{T}$ denote a decision tree that maps any data point x to a leaf node $t = \mathcal{T}(x)$ and assigns c_t as its prediction. $\mathcal{T}_L$ denotes the set of leaf nodes. A leaf node t represents a box in feature space $t_f = \{x' \in \mathbb{R}^p \mid t = \mathcal{T}(x)\}$. When this set intersects with the space of possible perturbations $S(x) = \{x + \delta \mid \delta \in S\}$, we say t is reachable and denote the set of reachable leafs using $\mathcal{T}_L^{S(x)} = \{t \in \mathcal{T}_L \mid t_f \cap S(x) \neq \emptyset\}$. We now present Robust Optimal Classification Trees (ROCT), which turns Equation 5.1 into a single minimization problem that can be solved using combinatorial optimization:

Theorem 4. *Robust learning (Equation 5.1) with 0-1 loss in the case of binary classification trees is equivalent to:*

$$\min_{\theta} \sum_{(x,y)\sim D} \left[\bigvee_{t\in\mathcal{T}_L^{S(x)}} c_t \neq y \right]$$

Proof. For 0-1 loss $L_{0\text{-}1}$, Equation 5.1 is equivalent to:

$$\min_{\theta} \sum_{(x,y)\sim D} \left(\max_{\delta\in S} L_{0\text{-}1}(\theta, x+\delta, y)\right)$$

Any perturbation in the inner maximization $\max_{\delta \in S}$ such that $\mathcal{T}(x) = \mathcal{T}(x+\delta)$ gives the same classification outcome for 0-1 loss. The maximization over all $\delta \in S$ can therefore be replaced by a maximization over all reachable leaf nodes $t \in \mathcal{T}_L^{S(x)}$. By definition, the 0-1 loss term is equivalent to the absolute difference $|c_t - y|$ of prediction c_t and label y, which gives:

$$\min_\theta \sum_{(x,y)\sim D} \left(\max_{t \in \mathcal{T}_L^{S(x)}} |c_t - y| \right)$$

The term $|c_t - y|$ takes value 1 when $c_t \neq y$ and 0 otherwise. When any of the reachable leaves $t \in \mathcal{T}_L^{S(x)}$ predict $c_t \neq y$, the inner maximization becomes 1. This is equivalent to the disjunction over $c_t \neq y$ for all reachable leaves:

$$\min_\theta \sum_{(x,y)\sim D} \left[\bigvee_{t \in \mathcal{T}_L^{S(x)}} c_t \neq y \right]$$

□

ROCT solves this formulation in one shot using discrete optimization solvers. We present 6 versions that vary in the kind of solver (MILP or MaxSAT) and type of variables used to represent splitting thresholds, see Table 5.2.

5.2.1 Attack Model

We assume the existence of a white-box adversary that can move all samples within a box-shaped region around each sample. This box-shaped region is defined by two vectors Δ^l and Δ^r from $\mathbb{R}^n$ specifying for each feature $i \in [1, n]$ how much i can be decreased and increased respectively, i.e., $S = \{\delta \in \mathbb{R}^n : \forall_{1 \leq i \leq n} \Delta_i^l \leq \delta_i \leq \Delta_i^r\}$. For the ease of our formulation, we scale all feature values to be in the range $[0, 1]$, which means that the values in Δ^l and Δ^r encode distance as a fraction of the feature range. While our encoding is more flexible, we only test on attack models where $\Delta^l = \Delta^r = (\epsilon, \ldots, \epsilon)$, encoding an L^∞ norm with ϵ perturbation radius. This allows us to easily evaluate performance against a variety of attacker strengths.

5.2.2 Intuition

We borrow much of the notation from OCT [8], summarized in Table 5.1. Figure 5.2 visualizes the variables in ROCT and Figure 5.3 shows an example of the constraints for a single sample and a tree of depth 1. In the regular learning setting where samples cannot be perturbed by an adversary, samples can only propagate to the left or right child of decision node. In the adversarial setting, samples can permute and are able to reach both the left and right sides, i.e. s_{im0} and s_{im1} can be true at the same time.

Given the attacker capabilities Δ^l and Δ^r, we create the constraints to set the variables s. To determine whether sample X_i can move left of the chosen split we can decrease its feature values as far as the attacker capabilities allow ($X_i - \Delta^l$) and see if it reaches the left side. Similarly to see if it reaches the right side we increase the feature values maximally ($X_i + \Delta^r$). We give two kinds of constraints for determining these s variables that differ in whether decision thresholds are represented by binary or continuous variables.

Symbol	Type	Definition
a_{jm}	variable	node m splits on feature j
b_{vm}	variable	node m's threshold is left/right of v
b'_m	variable	node m's continuous threshold value
c_t	variable	leaf node t predicts class 0 or 1
s_{im0}	variable	sample i can move left of node m
s_{im1}	variable	sample i can move right of node m
e_i	variable	sample i can be misclassified
X_{ij}	constant	value of data row i in feature j
y_i	constant	class label of data row i
Δ^l_j	constant	left perturbation range for feature j
Δ^r_j	constant	right perturbation range for feature j
n	constant	number of samples
p	constant	number of features
$A(t)$	set	ancestors of node t
$A_l(t)$	set	... with left branch on the path to t
$A_r(t)$	set	... with right branch on the path to t
S	set	all possible perturbations
$S(x)$	set	... applied to sample x
$\mathcal{T}_B$	set	all decision nodes
$\mathcal{T}_L$	set	all leaf nodes
$\mathcal{T}_L^{S(x)}$	set	... that intersect with $S(x)$
V_j	set	unique values in feature j

Table 5.1: Summary of the notation used throughout the paper.

Continuous Decision Thresholds

To select a threshold value an intuitive method is to create a continuous variable b_m for every decision node. We can then use this variable to determine the values s_{im0} and s_{im1} by checking whether $X_i - \Delta^l$ and $X_i + \Delta^r$ can reach the left and right side of the threshold respectively. We create the following constraints:

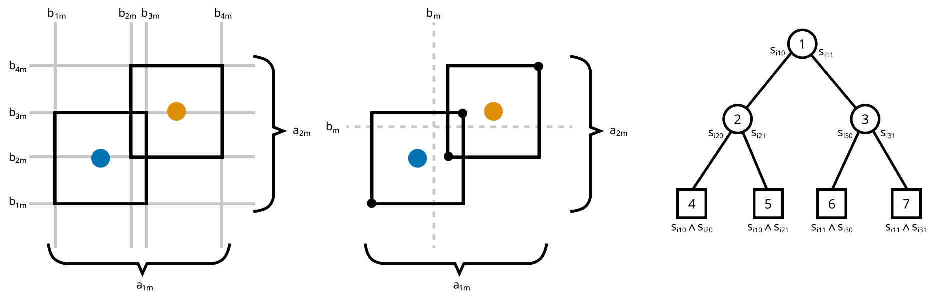
$$(\mathbf{X_i} - \mathbf{\Delta^l}) \cdot \mathbf{a_m} \leq b'_m \implies s_{im0}$$

$$(\mathbf{X_i} + \mathbf{\Delta^r}) \cdot \mathbf{a_m} > b'_m \implies s_{im1}$$

Since these constraints use a dot product with continuous variables it is not possible to implement this in MaxSAT. Another challenge comes with the second constraint being a strict inequality which is not directly supported in MILP. Like [8], we add a small value to the right hand side to turn it into a regular inequality.

Binary Decision Thresholds

We create a set of variables b_{vm} for each unique decision threshold value v, with v in ascending order. Instead of forcing one of them to `true`, we create an ordering in the variables



(a) Decision node *m* (binary) (b) Decision node *m* (continuous) (c) Path variables for sample *i*

Figure 5.2: Example of ROCT's formulation. For each decision node the *a* variables select a splitting feature and *b* select the threshold value. *b* can be defined as multiple binary (a) or a single continuous (b) variable. Using the *s* variables (c) ROCT traces all sample paths through the tree to the leaves and counts an error if any reachable leaf predicts the wrong class.

Method	Threshold formulation	Solver	Init. with GROOT
LSU-MaxSAT	binary	LSU (glucose 4.1)	no
RC2-MaxSAT	binary	RC2 (glucose 4.1)	no
Binary-MILP	binary	GUROBI 9	no
Binary-MILP-warm	binary	GUROBI 9	yes
MILP	continuous	GUROBI 9	no
MILP-warm	continuous	GUROBI 9	yes

Table 5.2: Summary of introduced methods, they differ in solver type and whether thresholds are formulated with binary or continuous variables. The 'warm' methods are initialized with the GROOT heuristic.

such that if one threshold variable is `true`, the larger variables also become `true`:

$$b_{vm} \implies b_{(v+1)m}$$

Intuitively if b_{vm} is set to `true` a sample with feature value v will be sent to the right of the split and when b_{vm} is `false` it will be sent to the left. A useful property of this constraint is that we only have to encode the local influence of a threshold variable b_{vm} on close-by data points, the rest is forced by the chain of constraints. For each feature j we determine what threshold values v^l and v^r correspond to $X_{ij} - \Delta_j^l$ and $X_{ij} + \Delta_j^r$ and check whether their b_{vm} values indicate that the sample can reach the left / right side:

$$a_{jm} \wedge \neg b_{v^l m} \implies s_{im0}$$

$$a_{jm} \wedge b_{v^r m} \implies s_{im1}$$

Selecting Features

Consider a single decision node m, such a decision node needs to decide a feature to split on. We create a binary variable a_{jm} for each feature j and force that exactly one of these variables can be equal to 1:

$$\sum_{j=1}^{p} a_{jm} = 1$$

This constraint can be relaxed to $\sum_{j=1}^{p} a_j \geq 1$ as selecting more than one feature can only make more s variables `true` and thus can only increase the number of errors.

Counting errors

We create a variable e_i for each sample i which is true when any reachable leaf $t \in \mathcal{T}_L^{S(x)}$ (see Theorem 1) predicts the other class. These leaves are found by following all paths a sample can take through the tree using the s_{im0} and s_{im1} variables. This is visualized in Figure 5.2c. Sample i can reach leaf t when the values $s_{im...}$ are `true` for all nodes m on the path to t, i.e. $\bigwedge_{m \in A_l(t)} s_{im0} \bigwedge_{m \in A_r(t)} s_{im1}$. Here $A_l(t)$ refers to the set of ancestors of leaf t of which we follow the path through its left child and $A_r(t)$ for child nodes on the right. When sample i can reach leaf t and its label does not match t's prediction ($y_i \neq c_t$), force e_i to `true`:

$$\bigwedge_{m \in A_l(t)} s_{im0} \bigwedge_{m \in A_r(t)} s_{im1} \wedge (c_t \neq y_i) \implies e_i$$

With one constraint per decision leaf and sample combination this determines the e values. To then turn all possible paths into predictions we need to assign a prediction label to each decision leaf. Each leaf t gets a variable c_t where `false` means class 0 and `true` means class 1.

Objective Function

Our goal is to minimize the equation from Theorem 1. This is equivalent to minimizing the sum of errors e_i ($i = 1...n$). We convert this MILP objective to MaxSAT by adding a soft constraint $\neg e_i$ for each sample and maximizing the number of correctly predicted samples:

$$\text{maximize} \quad \sum_{i=1}^{n} \neg e_i \quad \text{or} \quad \text{minimize} \quad \sum_{i=1}^{n} e_i$$

5.2.3 Complete Formulation

Below we give the full formulation for ROCT, in Table 5.1 we summarize the notation used. The equations can easily be formulated as MILP or MaxSAT instances, for MILP this was done with big-M constraints.

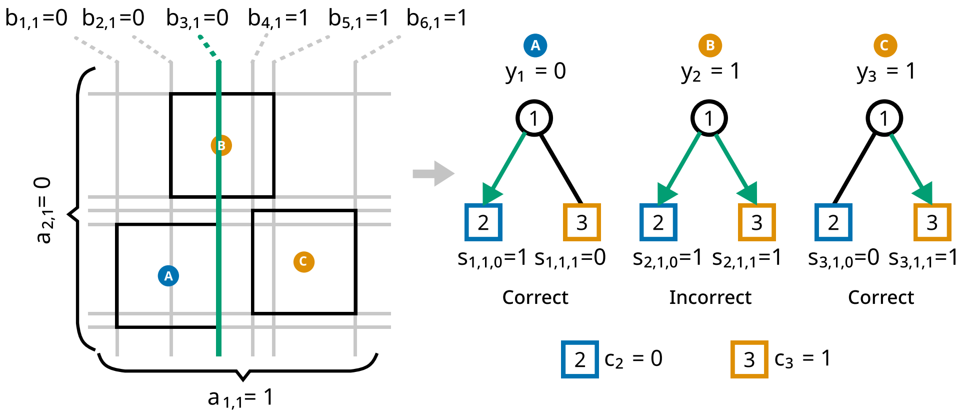


Figure 5.3: Example of a decision tree of depth 1 with the binary threshold formulation. Sample A and C get correctly classified since all their reachable leaves predict the correct label. Sample B reaches both leaves, since the left leaf predicts the wrong label, B gets misclassified.

$$\min. \sum_{i=1}^{n} e_i$$

subject to:

$$\sum_{j=1}^{p} a_{jm} = 1, \qquad \forall m \in \mathcal{T}_B$$

$$b_{vm} \Rightarrow b_{(v+1)m}, \qquad \forall m \in \mathcal{T}_B, v{=}1..|V_j|-1$$

$$\bigwedge_{m \in A_l(t)} s_{im0} \bigwedge_{m \in A_r(t)} s_{im1} \wedge [c_t{\neq}y_i] \Rightarrow e_i, \qquad \forall t \in \mathcal{T}_L, i{=}1..n$$

continuous threshold variables:

$$(\mathbf{X_i} - \Delta^{\mathbf{l}}) \cdot \mathbf{a_m} \leq b'_m \Rightarrow s_{im0} \qquad \forall m \in \mathcal{T}_B, i{=}1..n$$

$$(\mathbf{X_i} + \Delta^{\mathbf{r}}) \cdot \mathbf{a_m} > b'_m \Rightarrow s_{im1} \qquad \forall m \in \mathcal{T}_B, i{=}1..n$$

binary threshold variables:

$$a_{jm} \wedge \neg b_{v^l m} \Rightarrow s_{im0}, \qquad \forall m \in \mathcal{T}_B, i{=}1..n, j{=}1..p$$

$$a_{jm} \wedge b_{v^r m} \Rightarrow s_{im1}, \qquad \forall m \in \mathcal{T}_B, i{=}1..n, j{=}1..p$$

In both the continuous and binary threshold formulations the size of the instances is dominated by the constraints setting the s variables. In the continuous case this size is of complexity $\mathcal{O}(2^d n)$ where d is the depth of the tree and n the number of samples. For the binary threshold case the size complexity is $\mathcal{O}(2^d np)$ where p is the number of features. The solvers run in (worst-case) exponential time.

Example

For clarity we give a small example of a decision tree of depth 1 that we fit on 3 samples. In Figure 5.3, we show three data points $A{=}(0.2, 0.2), B{=}(0.5, 0.8), C{=}(0.8, 0.3)$ and all their

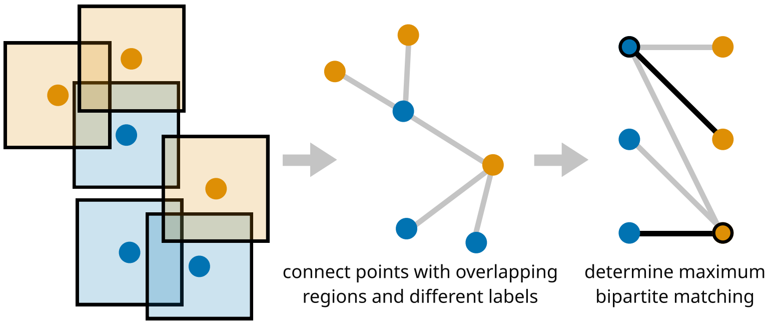

connect points with overlapping regions and different labels
determine maximum bipartite matching

Figure 5.4: Computing a bound on adversarial accuracy by maximum matching. The maximum matching and minimum vertex cover are shown in black. Since the matching has a cardinality of 2 it is impossible to misclassify fewer than 2 samples when accounting for perturbations.

feature values can be perturbed within a L^∞ norm radius 0.2. This results in feature 1 taking one of the 6 possible threshold values: $\{0.0, 0.3, 0.4, 0.6, 0.7, 1.0\}$ (due to bounding boxes). Suppose the solver selects feature 1 for decision node 1: $a_{1,1} = 1$ and $a_{2,1} = 0$. Suppose the solver selects the third threshold value: $b_{1,1} = b_{1,2} = b_{1,3} = 0$ and $b_{1,4} = b_{1,5} = b_{1,6} = 1$ (note this is a unary encoding). Due to the binary threshold constraints we then obtain:

$$a_{1,1} \wedge \neg b_{1,1} \Rightarrow s_{1,1,0}, \quad 1 \wedge \neg 0 \Rightarrow s_{1,1,0}{=}1$$

Thus, the first data point (A) can move to the left of decision node 1, since $b_{1,1} = 0$. From Figure 5.3, we see that $b_{1,1} = 0$ implies the decision threshold is to the right of the lower bound of the bounding box for point A. Hence indeed, it should be able to move left. Similarly for the upper bound:

$$a_{1,1} \wedge b_{3,1} \Rightarrow s_{1,1,1}, \quad 1 \wedge 0 \Rightarrow s_{1,1,1}{\in}\{0,1\}$$

Thus, since $b_{3,1} = 0$, the constraints pose no restriction on whether point A can move to right of decision node 1. The correct behavior (A cannot move to the right) is forced by the objective function, which can only become worse by setting $s_{1,1,1} = 1$. The remaining s variables become:

$$a_{1,1} \wedge \neg b_{2,1} \Rightarrow s_{2,1,0}, \quad 1 \wedge \neg 0 \Rightarrow s_{2,1,0}{=}1$$

$$a_{1,1} \wedge b_{5,1} \Rightarrow s_{2,1,1}, \quad 1 \wedge 1 \Rightarrow s_{2,1,1}{=}1$$

$$a_{1,1} \wedge b_{6,1} \Rightarrow s_{3,1,1}, \quad 1 \wedge 1 \Rightarrow s_{3,1,1}{=}1$$

$s_{3,1,0}$ remains unconstrained and can therefore be set to 0 by the solver. Since $c_1 = y_1$ and $c_3 = y_3$, e_1 and e_3 are unconstrained and minimized to 0 by the solver, and since $s_{2,0} = s_{2,1} = 1$ the constraints force $e_2 = 1$. The second sample is hence misclassified (it reaches at least one leaf with a prediction value different than its label). Note that, although the thresholds in Figure 5.3 are always exactly on the perturbation ranges of a sample, we post-process these to maximize the margin.

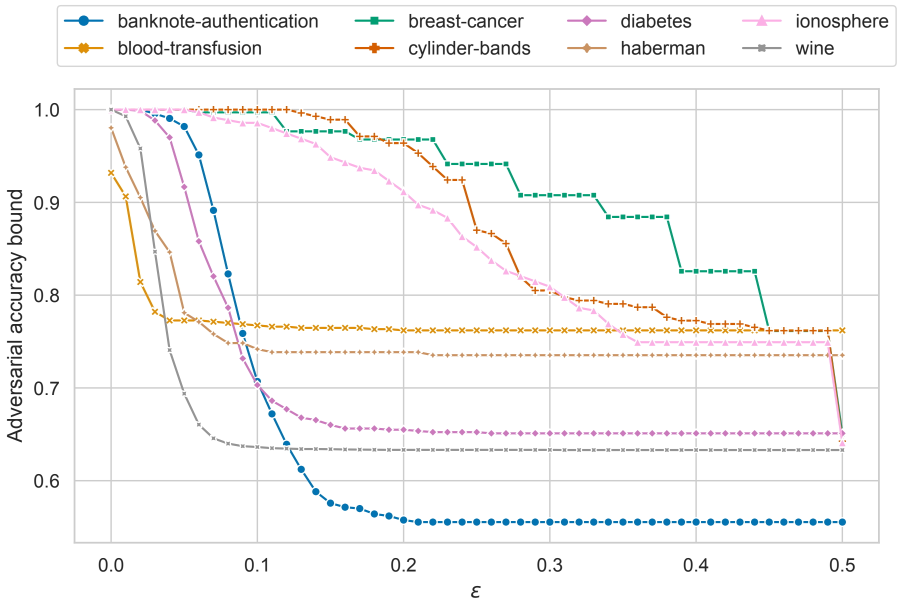


Figure 5.5: Varying the L^∞ perturbation radius ϵ and computing the adversarial accuracy bound. Datasets are affected differently, e.g. ϵ=0.1 has no effect on cylinder-bands while the bound for blood-transfusion shows that it is not possible to score better than constantly predicting its majority class.

5.3 Upper Bound on Adversarial Accuracy

In a regular learning setting with stationary samples one strives for a predictive accuracy of 100%. As long as there are no data points with different labels but same coordinates achieving this score is theoretically possible. However, we realize that in the adversarial setting a perfect classifier cannot always score 100% accuracy as samples can be perturbed. We present a method to compute the upper bound on adversarial accuracy using a bipartite matching that can be computed regardless of what model is used. We use this bound to choose better ϵ values for our experiments. It also lets us compare the scores of optimal decision trees to a score that is theoretically achievable by perfect classifiers. Such a matching approach was also used in [15] to train robust kNN classifiers.

Theorem 5. *The maximum cardinality bipartite matching between samples with overlapping perturbation range and different labels* $\{(i, j) : S_i \cap S_j \neq \emptyset \wedge y_i \neq y_j\}$ *gives an upper bound to the adversarial accuracy achievable by any model for binary decision problems.*

Proof. The reduction to maximum bipartite matching is based on the realization that when the perturbation ranges of two samples with different labels overlap it is not possible to predict both of these samples correctly. A visual explanation is given in Figure 5.4. Formally, given a classifier C that maps samples to a class 0 or 1, a sample i can only be correctly predicted against an adversary if its entire perturbation range S_i is correctly

Dataset (OpenML)	n	p	Maj.
haberman (1)	306	3	.735
blood-transfusion-service-center (1)	748	4	.762
cylinder-bands (2)	277	37	.643
diabetes (1)	768	8	.651
ionosphere (1)	351	34	.641
banknote-authentication (1)	1372	4	.555
breast-w (1)	683	9	.650
wine_quality (1)	6497	11	.633

Table 5.3: Overview of datasets used in the experiments. Number of samples, features and ratio of majority class samples.

predicted:

$$\forall x \in S_i : C(x) = y_i \tag{5.2}$$

Now given a sample j of a different class (e.g. $y_i = 0$ and $y_1 = 1$) that has an overlapping perturbation range such that $S_i \cap S_j \neq \emptyset$, it is clear that Equation 5.2 cannot simultaneously hold for both samples. We create a bipartite graph $G = (V_0, V_1, E)$ with $V_0 = \{i : y_i = 0\}$ and $V_1 = \{i : y_i = 1\}$, i.e., vertices representing samples of class 0 on one side and class 1 on the other. We then connect two vertices with an edge if their perturbation ranges overlap and their labels are different: $E = \{(i,j) : S_i \cap S_j \neq \emptyset \wedge y_i \neq y_j\}$.

To obtain the upper bound, we consider the minimum vertex cover V' from G. By removing all vertices / samples in V', none of the remaining samples can be transformed to have identical feature values with a sample from the opposite class. A perfect classifier $C'(x)$ would therefore assign these rows their correct class values and an attacker will not be able to influence the score of this classifier. It is not possible to misclassify fewer samples than the cardinality of the minimum vertex cover V' since removing any vertex from it will add at least one edge $e \in \{(i,j) : S_i \cap S_j \neq \emptyset \wedge y_i \neq y_j\}$ which will cause an additional misclassification. By König's theorem such a minimum cover in a bipartite graph is equivalent to a maximum matching. Therefore we can use a maximum matching solver to compute an upper bound on the adversarial accuracy. □

5.3.1 Improving Experiment Design

In previous works [4, 5] attacker capabilities were arbitrarily chosen but this limits the value of algorithm comparisons, shown in Figure 5.5. In this figure we vary the L^∞ radius ϵ by which an adversary can perturb samples. Particularly, if this value is chosen too large, the best possible model is a trivial one that constantly predict the majority class. If ϵ is chosen too small, the adversary has no effect on the learning problem.

To improve the design of our experiments we propose to choose values for ϵ along these curves that cause the adversarial accuracy bound to be non-trivial. In our experiments we choose three ϵ values for each dataset such that their values corresponds to an adversarial accuracy bound that is at 25%-50%-75% of the range. When choosing ϵ at 100%

of the range, the bound is equal to the ratio of the majority class samples, i.e. predicting only that class.

5.4 Results

To demonstrate the effectiveness of ROCT we compare it to the state-of-the-art robust tree learning algorithms TREANT and GROOT, and to the regular decision trees from scikit-learn [16]. First we run the algorithms on an artificial XOR dataset to show that the heuristics can theoretically learn arbitrarily bad trees, see Figure 5.1. Then to compare the practical performance we run the algorithms on eight popular datasets [3, 5] and varying perturbation radii (ϵ). All of our experiments ran on 15 Intel Xeon CPU cores and 72 GB of RAM total, where each algorithm ran on a single core. These datasets are used in many of the existing works to compare robust tree learning algorithms. The datasets are summarized in Table 5.3 and are available on OpenML[2].

5.4.1 Predictive Performance on Real Data

To demonstrate the practical performance of ROCT we compared the scores of ROCT, GROOT and TREANT on eight datasets. For each dataset we used an 80%-20% train-test split. To limit overfitting it is typical to constrain the maximum depth of the decision tree. To this end we select the best value for the maximum depth hyperparameter using 3-fold stratified cross validation on the training set. In each run, every algorithm gets 30 minutes to fit. For MILP, binary-MILP and LSU-MaxSAT this means that we stop the solver and retrieve its best solution at that time. The methods GROOT, TREANT and RC2-MaxSAT cannot return a solution when interrupted. Therefore when these algorithms exceed the timeout we use a dummy classifier that predicts a constant value. As the dual of the MILP-based formulations is hard to solve, we focus the solver on the primal problem. The final adversarial accuracy scores were determined by testing for each sample whether a sample with a different label intersects its perturbation range.

Table 5.4 shows the aggregated results over these 8 datasets, 5.5 contains the individial test scores for a selection of the compared methods. The overall best scores were achieved with the MILP-warm method which is the MILP formulation with continous variables for thresholds and is warm started with the tree produced by GROOT. The LSU-MaxSAT method also performed well and runs without reliance on trees trained with GROOT. TREANT's scores were lower than expected which can be attributed to the number of time outs.

5.4.2 Runtime

An advantage of using optimization solvers for training robust decision trees is that most solvers can be early stopped to output a valid tree. In figure 5.6 we plotted the mean training scores over all datasets for trees of depth 3 of the solvers that can be stopped. We see that all algorithms converge to nearly the same value given enough time. Moreover we find that LSU-MaxSAT quickly achieves good scores where it takes MILP-warm and Binary-MILP-warm approximately 10 and 100 seconds to catch up. The MILP-based meth-

[2]http://www.openml.org

Algorithm	Mean adv. accuracy	Mean rank	Wins
Decision Tree	.388 ± .055	8.917 ± .083	0
TREANT	.692 ± .013	5.167 ± .604	7
Binary-MILP	.714 ± .013	3.958 ± .576	10
MILP	.720 ± .015	2.917 ± .454	12
RC2-MaxSAT	.724 ± .014	2.667 ± .393	10
GROOT	.726 ± .015	2.375 ± .450	16
Binary-MILP-warm	.726 ± .015	2.083 ± .399	16
LSU-MaxSAT	.729 ± .014	2.125 ± .303	13
MILP-warm	**.735** ± .015	**1.583** ± .225	**17**

Table 5.4: Aggregate test scores over 8 datasets, means are shown with standard error. All methods trained for 30 minutes and selected their depth using 3-fold cross validation.

ods that were not warm started with GROOT took approximately 1000 seconds to catch up with LSU-MaxSAT.

5.4.3 Optimality

Existing robust decision tree learning algorithms such as TREANT and GROOT have no performance guarantees. Using the LSU-MaxSAT solver we can find trees and prove their optimality on the training set which allows us to compare the scores of the heuristics with these optimal scores. In Figure 5.7 we plot the approximation ratios of GROOT trees after 2 hours of training. Although LSU-MaxSAT was not able to prove optimality for many datasets after a depth of 2 we can still see that GROOT scores close to optimal. All but one tree scores within a ratio of 0.92 with only one case having a ratio of approximately 0.87. We also plot the ratio between our upper bound and optimal trees. Interestingly, optimal trees of depths 1 and 2 already score close to the upper bounds in some cases.

Dataset	ϵ	Dec. Tree	MILP	GROOT	LSU	MILP warm
banknote-authentication	.07	.665	.742	.775	.796	**.822**
	.09	.589	.669	.684	**.724**	.720
	.11	.491	.625	.640	**.644**	.629
blood-transfusion-service-center	.01	.687	.747	.720	**.760**	.747
	.02	.647	.727	.727	**.767**	**.767**
	.03	.627	**.767**	**.767**	.760	**.767**
breast-cancer	.28	.095	**.869**	**.869**	**.869**	**.869**
	.39	.073	**.818**	**.818**	**.818**	**.818**
	.45	.073	**.774**	**.774**	**.774**	**.774**
cylinder-bands	.23	.000	**.732**	**.732**	.714	.714
	.28	.000	.679	.643	.679	**.750**
	.45	.000	.643	.643	**.679**	.643
diabetes	.05	.455	**.649**	**.649**	**.649**	**.649**
	.07	.364	**.649**	**.649**	**.649**	**.649**
	.09	.286	**.649**	**.649**	**.649**	**.649**
haberman	.02	.726	**.742**	.726	**.742**	**.742**
	.03	.726	**.742**	**.742**	**.742**	**.742**
	.05	.677	**.742**	**.742**	**.742**	**.742**
ionosphere	.2	.310	.817	**.845**	.817	**.845**
	.28	.169	.817	**.845**	.817	**.845**
	.36	.042	**.775**	**.775**	**.775**	**.775**
wine	.02	.602	.638	**.680**	.661	.674
	.03	.541	.633	**.662**	.639	**.662**
	.04	.472	.633	**.659**	.635	**.659**

Table 5.5: Individual test scores for each dataset and ϵ combination. Best scores are marked in bold.

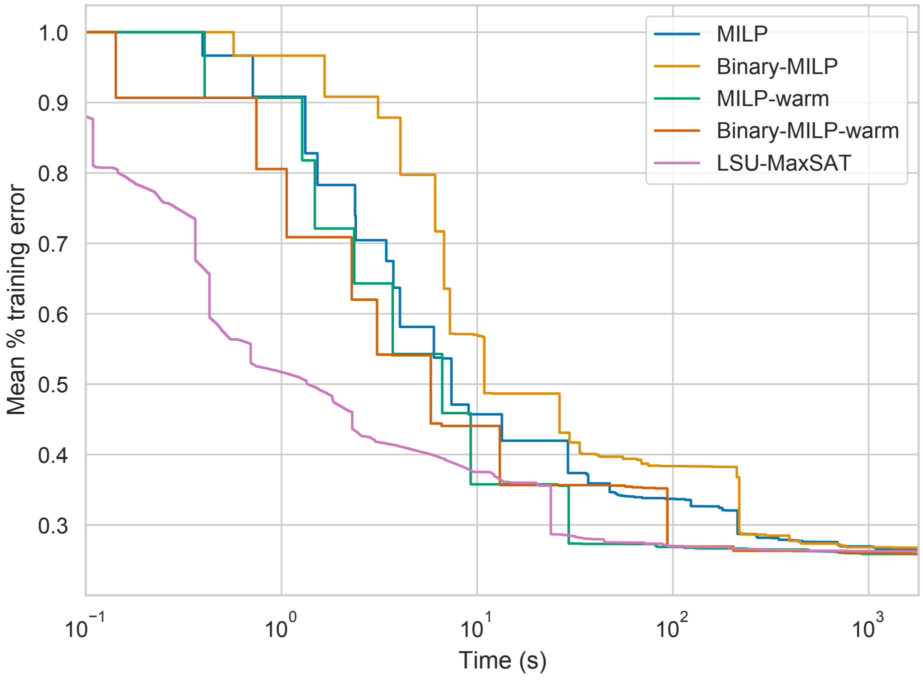


Figure 5.6: Mean percentage of misclassified training samples of all 8 datasets over time for trees of depth 3. The ranges represent one standard error. LSU-MaxSAT is faster at first but after 30 minutes the other methods catch up.

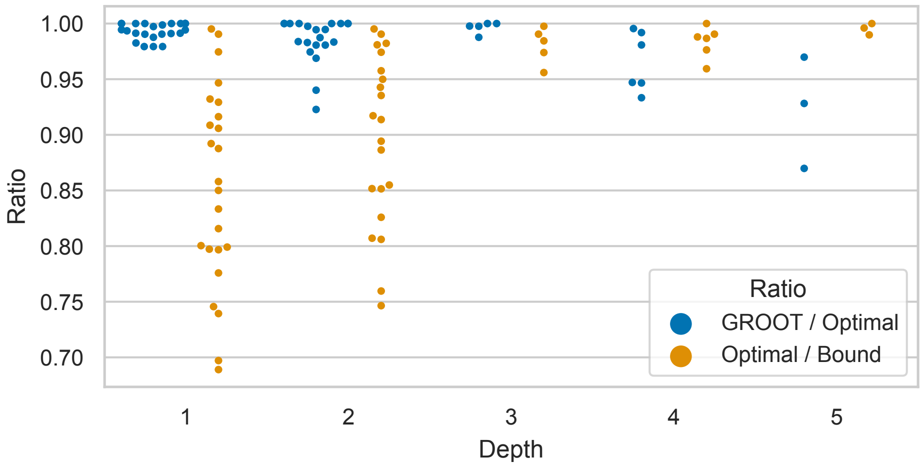


Figure 5.7: Ratios of training adversarial accuracy scores between GROOT vs LSU-MaxSAT's optimal trees and optimal trees vs our bound (Theorem 2). In most cases GROOT performs within 5% of optimal. In some cases trees of depth 1 or 2 already score as well as the upper bound.

5.5 Conclusions

In this work we propose ROCT, a new solver based method for fitting robust decision trees against adversarial examples. Where existing methods for fitting robust decision trees can perform arbitrarily poorly in theory, ROCT fits the optimal tree given enough time. Important for the computational efficiency of ROCT is the insight and proof that the min-max adversarial training procedure can be computed in one shot for decision trees (Theorem 1). We compared ROCT to existing methods on 8 datasets and found that given 30 minutes of runtime ROCT improved upon the state-of-the-art. Moreover, although greedy methods have been compared to each other in earlier works, we demonstrate for the first time that the state-of-the-art actually performs close to optimal. We also presented a new upper bound for adversarial accuracy that can be computed efficiently using maximum bipartite matching (Theorem 2).

Although ROCT was frequently able to find an optimal solution and shows competitive testing performance, the choice of tree depth strongly influences runtime. Optimality could only be proven for most datasets up to a depth of 2 and for some until depth 4. Additionally, the size of ROCT's formulation grows linearly in terms of the number of unique feature values of the training dataset which results in an exponential increase in runtime. For small datasets of up to a few 1000 samples and tens of features ROCT is likely to improve performance over state-of-the-art greedy methods. Overall, ROCT can increase the performance of state-of-the-art heuristic methods and, due to its optimal nature and new upper bound, provide insight into the difficulty of robust learning.

In the future, we will investigate realistic use cases of adversarial learning in security such as fraud / intrusion / malware detection. Such use cases might have more complicated attackers that require non-box-shaped threat models. We expect our upper-bound method to be a useful tool in determining the sensibility of adversarial learning problems and for robust feature selection.

Bibliography

[1] D. Vos and S. Verwer, *Robust optimal classification trees against adversarial examples,* Proceedings of the AAAI Conference on Artificial Intelligence **36**, 8520 (2022).

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing properties of neural networks,* arXiv preprint arXiv:1312.6199 (2013).

[3] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, *Robust decision trees against adversarial examples,* arXiv preprint arXiv:1902.10660 (2019).

[4] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, *Treant: training evasion-aware decision trees,* Data Mining and Knowledge Discovery **34**, 1390 (2020).

[5] D. Vos and S. Verwer, *Efficient training of robust decision trees against adversarial examples,* arXiv preprint arXiv:2012.10438 (2020).

[6] E. Carrizosa, C. Molero-Río, and D. R. Morales, *Mathematical optimization in classification and regression trees,* TOP , 1 (2021).

[7] H. Laurent and R. L. Rivest, *Constructing optimal binary decision trees is np-complete,* Information processing letters **5**, 15 (1976).

[8] D. Bertsimas and J. Dunn, *Optimal classification trees,* Machine Learning **106**, 1039 (2017).

[9] S. Verwer and Y. Zhang, *Learning decision trees with flexible constraints and objectives using integer optimization,* in *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Springer, 2017) pp. 94–103.

[10] N. Narodytska, A. Ignatiev, F. Pereira, J. Marques-Silva, and I. RAS, *Learning optimal decision trees with sat.* in *IJCAI* (2018) pp. 1362–1368.

[11] F. Avellaneda, *Efficient inference of optimal decision trees,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34 (2020) pp. 3195–3202.

[12] H. Verhaeghe, S. Nijssen, G. Pesant, C.-G. Quimper, and P. Schaus, *Learning optimal decision trees using constraint programming,* Constraints **25**, 226 (2020).

[13] D. I. Diochnos, S. Mahloujifar, and M. Mahmoody, *Adversarial risk and robustness: General definitions and implications for the uniform distribution,* arXiv preprint arXiv:1810.12272 (2018).

[14] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards deep learning models resistant to adversarial attacks,* arXiv preprint arXiv:1706.06083 (2017).

[15] Y. Wang, S. Jha, and K. Chaudhuri, *Analyzing the robustness of nearest neighbors to adversarial examples,* in *International Conference on Machine Learning* (PMLR, 2018) pp. 5133–5142.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python,* Journal of Machine Learning Research **12**, 2825 (2011).

6 Robust Leaf Relabeling

Decision trees are popular models for their interpretation properties and their success in ensemble models for structured data. However, common decision tree learning algorithms produce models that suffer from adversarial examples. Recent work on robust decision tree learning mitigates this issue by taking adversarial perturbations into account during training. While these methods generate robust shallow trees, their relative quality reduces when training deeper trees due the methods being greedy. In this work we propose robust relabeling, a post-learning procedure that optimally changes the prediction labels of decision tree leaves to maximize adversarial robustness. We show this can be achieved in polynomial time in terms of the number of samples and leaves. Our results on 10 datasets show a significant improvement in adversarial accuracy both for single decision trees and tree ensembles. Decision trees and random forests trained with a state-of-the-art robust learning algorithm also benefited from robust relabeling.

This chapter is based on *D. Vos and S. Verwer. Adversarially Robust Decision Tree Relabeling, ECML-PKDD 2022* [1].

6.1 Introduction

With the increasing interest in trustworthy machine learning, decision trees have become important models [2]. Due to their simple structure humans can interpret the behavior of size-limited decision trees. Additionally, decision trees are popular for use within ensemble models where random forests [3] and particularly gradient boosting ensembles [4–7] achieve top performance on prediction tasks with tabular data. However, decision trees are optimized without considering robustness which results in models that misclassify many data points after adding tiny perturbations [8, 9], i.e. adversarial examples. Therefore we are interested in training tree-based models that correctly predict data points not only at their original coordinates but also in a radius around these coordinates.

Recent work has proposed decision tree learning algorithms that take adversarial perturbations into account during training to improve adversarial robustness [10–12]. These methods significantly improved robustness for shallow decision trees, but lacked performance for deeper trees due to their greedy nature. Optimal methods for robust decision tree learning [13, 14] have also been proposed but they use combinatorial optimization solvers which makes them scale poorly in terms of both tree depth and data size. Adversarial pruning [15, 16] is a method that pre-processes datasets by removing a minimal number of samples to make the dataset well-separated. While this method helps ignore samples that will only worsen robustness when predicted correctly, the learning algorithm is unchanged so the resulting models still suffer from adversarial examples. It is important to be able to train deeper robust trees as shallow trees can significantly underfit the data. Particularly in random forests where we aim to ensemble unbiased models [17] we need to be able to train very deep trees.

To improve the performance of robust decision trees we propose Robust Relabeling[1]. This post-learning procedure optimally changes the prediction labels of the decision tree leaves to maximize accuracy against adversarial examples. We assume that the user specifies an arbitrary region around each sample that represents the set of all possible perturbations of the sample. Then, we only consider a sample to be correctly predicted under adversarial attacks if there is no way for an attacker to perturb the sample such that the prediction is different from the label. We prove that in binary classification the optimal robust relabeling is induced by the minimum vertex cover of a bipartite graph. This property allows us to compute the relabeling in polynomial time in terms of the number of samples and leaves.

We compare the classification performance of decision trees and tree ensemble models on 10 datasets from the UCI Machine Learning Repository [18] and find that robust relabeling improves the average adversarial accuracy for all models. We also evaluate the performance when relabeling robust decision trees trained with a state-of-the-art method GROOT [12]. The resulting models improve adversarial accuracy compared to the default GROOT models by up to 20%. Additionally, we study the effects of standard Cost Complexity Pruning against robust relabeling. Both methods reduce the size of the learned decision trees and can improve both regular accuracy and adversarial accuracy compared to unpruned models. While, Cost Complexity Pruning performs better on regular accuracy, robust relabeling results in better adversarial accuracy.

[1]`https://github.com/tudelft-cda-lab/robust-relabeling`

6.2 Background Information

6.2.1 Robust Decision Tree Learning

In the field of robust decision trees, we usually assume that an adversary modifies our data points at test time to cause misclassifications. Then, we aim to train a decision tree that is maximally robust to such modifications. The type of modifications that we allow the adversary to make strongly influences the learned trees. In this work, we consider an adversary that can make arbitrary changes to each test data point i within a radius ϵ of the original point. In line with previous works [10, 12] we measure this distance with the L^∞ norm. Therefore the set of all possible perturbations applied to data point i is $S(i) = \{x + \delta : \|\delta\|_\infty \leq \epsilon\}$. For a decision tree $\mathcal{T}$ it is especially important to know what leaves $\mathcal{T}_L$ sample i can reach after applying perturbations, we refer to this set as $\mathcal{T}_L^{S(i)}$.

6.2.2 Minimum Vertex Covers and Robustness

To the best of our knowledge, Wang et al. [16] first published that for any given dataset D, there can be pairs of samples that can never be simultaneously correctly predicted against adversarial examples. For example, when considering perturbations within some radius ϵ, two samples with different labels that are within distance 2ϵ cannot both be correctly predicted when accounting for these perturbations. Given this fact, one can create a graph G with each vertex representing a sample and connect all such pairs. When we compute the minimum vertex cover of this graph, we find the minimum number of data points C to remove from D such that $D \setminus C$ can be correctly predicted. Although $D \setminus C$ *can* be correctly predicted, non-robust learning algorithms can and will still learn models that suffer from adversarial examples, e.g. because decision planes are placed too close to the remaining data points.

Wang et al. [16] used this minimum vertex cover idea by removing C from the training data in order to learn a robust nearest neighbor classifier. Adversarial pruning [15] uses a similar method to train nearest neighbor models, decision trees, and tree ensembles from $D \setminus C$. In ROCT (Chapter 5), we also used a minimum vertex cover to compute an upper bound on adversarial accuracy, which improved the time needed to train optimal robust decision trees and helped us choose experiment parameters.

6.2.3 Relabeling and Pruning Decision Trees

Improving the quality of decision trees with respect to some metrics by changing their leaf predictions is not a new idea. Many pruning algorithms have been proposed that remove parts of the decision tree to improve generalization. For example, Cost Complexity Pruning [19] is a widely used method that merges leaves when this improves the trade-off between the size of the tree and its predictive performance. Similarly, ideas to relabel decision trees have been used to improve performance for objectives such as fairness [20] and monotonicity [21]. Such metrics are not aligned with the objective that is optimized during training. To the best of our knowledge, we are the first to propose using leaf relabeling to improve adversarial robustness. Since relabeling methods never add new leaves, they can be seen as pruning methods since they reduce the size of the trees (after merging leaves that have the same label).

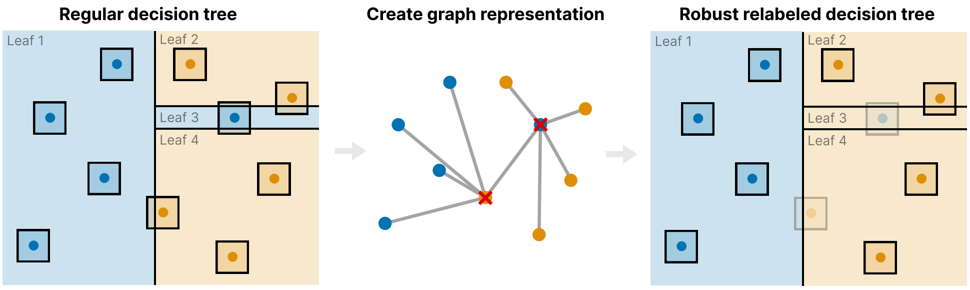


Figure 6.1: Example of the robust relabeling procedure applied to a decision tree that suffers from adversarial examples. We first create a bipartite graph that connects samples with different labels that can reach the same leaf with perturbations. After removing the samples corresponding to the graph's minimum vertex cover we can relabel the decision tree to correctly predict the remaining samples. The resulting labeling is maximally robust to adversarial perturbations.

6.3 Robust Relabeling

Since regular decision trees and ensembles suffer from adversarial examples we are interested in post-processing the learned models to improve their robustness. In this work, we propose 'robust relabeling' (Algorithm 6) a method that keeps the decision tree structure intact but changes the predictions in the leaves to maximize adversarial accuracy. Robust relabeling is closely related to earlier works that determine minimum vertex covers to improve robustness [13, 15, 16]. In these works, the authors leverage the fact that samples with overlapping perturbation ranges and different labels can never be simultaneously classified correctly under optimal adversarial perturbations. We notice that in decision trees, two samples cannot be simultaneously classified correctly under optimal adversarial perturbations when they both reach the same leaf. Using this property we can find the smallest set of samples to remove from the dataset such that all remaining samples *can* be classified correctly under perturbations. These samples then induce a labeling of the decision tree that correctly classifies the largest possible set of samples against adversarial perturbations.

To robustly relabel a decision tree $\mathcal{T}$ we create a bipartite graph $G = (L, R, E)$ where L represents the set of samples with label $y_i = 0$ and R the set of samples with label $y_j = 1$. The set of edges E is defined by connecting all pairs of samples (i, j) that have different labels $y_i \neq y_j$ and overlapping perturbation ranges $\mathcal{T}_L^{S(i)} \cap \mathcal{T}_L^{S(j)} \neq \emptyset$. Here $S(i)$ is the set of all possible perturbations applied to data point i and $\mathcal{T}_L^{S(i)}$ is the set of leaves that are reached by $S(i)$. We find the minimum vertex cover C of G and remove the samples represented by C from the dataset. We can then relabel the decision tree to classify all remaining samples correctly even under optimal adversarial attacks.

In this paper, we consider only the case where $S(i)$ describes an L^∞ radius around each data point, as this is common in research on robust decision trees. However, our proof does not make use of this fact and robust relabeling can easily be extended to other attack models such as different l-norms or arbitrary sets of perturbations.

Theorem 6. *The optimal adversarially robust relabeling for decision tree leaves $\mathcal{T}_L$ is determined by the minimum vertex cover of the bipartite graph where samples i, j with different*

Algorithm 6 Robust relabeling decision tree

```
Input: dataset $X$ ($n$ samples, $m$ features), labels $y$, tree leaves $\mathcal{T}_L$
 1: $L \leftarrow \{i \mid y_i = 0\}$                                              ▷ $\mathcal{O}(n)$
 2: $R \leftarrow \{i \mid y_i = 1\}$                                              ▷ $\mathcal{O}(n)$
 3: $E \leftarrow \{(u,v) \mid u \in L, v \in R, \mathcal{T}_L^{S(u)} \cap \mathcal{T}_L^{S(v)} \neq \emptyset\}$    ▷ $\mathcal{O}(nm|\mathcal{T}_L| + n^2|\mathcal{T}_L|)$
 4: $M \leftarrow$ MAXIMUM_MATCHING$(L,R,E)$                                       ▷ $\mathcal{O}(n^{2.5})$
 5: $C \leftarrow$ KŐNIG'S_THEOREM$(M,L,R,E)$                                      ▷ $\mathcal{O}(n)$
 6: for $t \in \mathcal{T}_L$ do                                                   ▷ $\mathcal{O}(n|\mathcal{T}_L|)$
 7:     if $\{i \in L \mid t \in \mathcal{T}_L^{S(i)}\} \neq \emptyset$ then
 8:         $c_t \leftarrow 0$
 9:     else
10:         $c_t \leftarrow 1$
11:     end if
12: end for
```

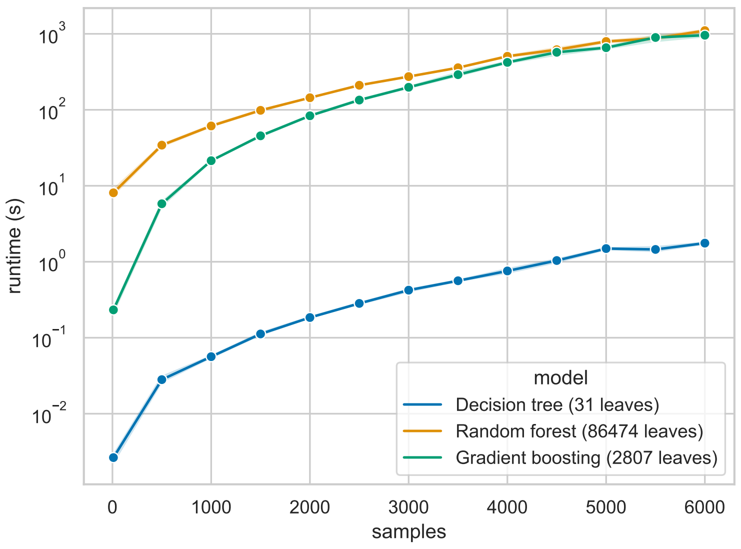
labels $y_i \neq y_j$ are represented by vertices that share an edge when their perturbations can reach any same leaf ($\mathcal{T}_L^{S(i)} \cap \mathcal{T}_L^{S(j)} \neq \emptyset$).

Proof. For a sample i to be correctly classified by a decision tree $\mathcal{T}$, all leaves $\mathcal{T}_L^{S(i)}$ reachable by adversarial perturbations applied to X_i need to predict the correct label, i.e. $\forall t \in \mathcal{T}_L^{S(i)}, c_t = y_i$ (otherwise an adversarial example exists). Given two samples i,j with different labels $y_i \neq y_j$ and overlapping sets of reachable leaves by perturbations $\mathcal{T}_L^{S(i)} \cap \mathcal{T}_L^{S(j)} \neq \emptyset$ these samples cannot be correctly robustly predicted as there exists a leaf t that is in both sets $\mathcal{T}_L^{S(i)}$ and $\mathcal{T}_L^{S(j)}$ and that misclassifies one of the samples. Create the bipartite graph $G = (L,R,E)$ where $L = \{i \mid y_i = 0\}$, $R = \{i \mid y_i = 1\}$ and $E = \{(u,v) \mid u \in L, v \in R, \mathcal{T}_L^{S(u)} \cap \mathcal{T}_L^{S(v)} \neq \emptyset\}$. By removing the minimum vertex cover C from G, we are left with the largest graph $G' = (L \setminus C, R \setminus C, \emptyset)$ for which no edges remain. Since none of the remaining vertices (representing samples) share an edge, we are able to set $\forall t \in \mathcal{T}_L^{S(i)}, \forall i \in (L' \cup R') : c_t = y_i$, so all remaining samples get correctly robustly classified. Since C is of minimum cardinality, the induced relabeling maximizes the adversarial accuracy. □

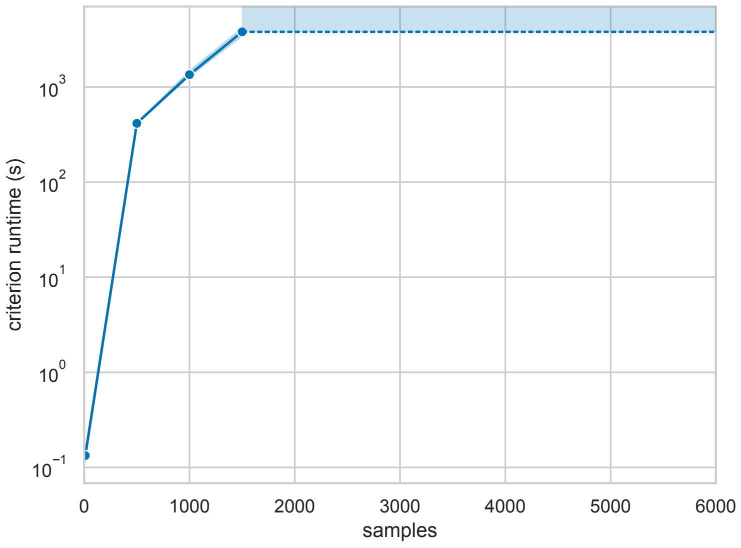
Where a naive relabeling algorithm would take exponential time to enumerate all $2^{|\mathcal{T}_L|}$ labelings, the above relabeling procedure runs in polynomial time in terms of the dataset size ($n \times m$ matrix) and the number of leaves $|\mathcal{T}_L|$. When building the graph G the runtime is dominated by computing the edges which takes $\mathcal{O}(nm|\mathcal{T}_L| + n^2|\mathcal{T}_L|)$ time. This is because we first build a mapping for each sample i to their reachable leaves $\mathcal{T}_L^S(i)$ in $\mathcal{O}(nm|\mathcal{T}_L|)$ time, then compute samples that reach any same leaf in $\mathcal{O}(n^2|\mathcal{T}_L|)$ time. Given the bipartite graph G we use the Hopcroft-Karp algorithm [22] to compute a maximum matching in $\mathcal{O}(n^{2.5})$ time and convert this in linear time into a minimum vertex cover using Kőnig's theorem. Combining all steps, robust relabeling runs in worst-case $\mathcal{O}(nm|\mathcal{T}_L| + n^2|\mathcal{T}_L| + n^{2.5})$ time.

6.3.1 Robust Relabeling as Splitting Criterion

While the robust relabeling procedure described before provides an intuitive use case as a post-processing step for decision tree learners, we can also use it to select splits during



(a) Robust relabeling



(b) Relabeling criterion tree

Figure 6.2: Runtime of robust relabeling and relabeling criterion trees on samples of the Wine dataset. While decision tree relabeling runs within seconds, relabeling ensembles takes longer due to the number of trees and the increase in tree size. The runtime for relabeling criterion trees quickly increases with the number of samples.

Table 6.1: Summary of datasets used. Features are scaled to $[0,1]$, so the L^∞ perturbation radius ϵ represents a fraction of each feature's range.

Dataset	ϵ	Samples	Features	Majority class
Banknote-authentication	.05	1,372	4	.56
Breast-cancer-diagnostic	.05	569	30	.63
Breast-cancer	.1	683	9	.65
Connectionist-bench-sonar	.05	208	60	.53
Ionosphere	.05	351	34	.64
Parkinsons	.05	195	22	.75
Pima-Indians-diabetes	.01	768	8	.65
Qsar-biodegradation	.05	1,055	41	.66
Spectf-heart	.005	349	44	.73
Wine	.025	6,497	11	.63

learning. In greedy decision tree learning, the learner finds a locally optimal split, partitions the samples into a left and right set (including perturbed samples), and continues this process recursively. While this approach finds an optimal split for the top decision node, the detrimental effect of choosing splits greedily increases with the depth of the tree. We will use the robust relabeling procedure to try to reduce the impact of greedily perturbing samples. To do this, we can consider all samples each time we score a split and use the cardinality of the maximum matching M as a splitting criterion. By choosing splits that minimize this criterion, we are then directly optimizing the adversarial accuracy of the decision tree. The pseudo-code for this algorithm is given in the appendix[2]. We will refer to this method as relabeling criterion trees.

6.3.2 Runtime Comparison

We compare the runtimes of robust relabeling and relabeling criterion trees on different sample sizes of the Wine dataset in Figure 6.2. Robust relabeling decision trees runs in a matter of seconds since the number of trees is small. In tree ensembles with 100 trees to relabel and many more leaves, the runtime quickly increases. We find that relabeling criterion trees take more than an hour to train on 2000 samples, and training on larger sample sizes quickly becomes infeasible.

In this work all experiments ran without parallelism on a laptop with 16GB of RAM and a 2 GHz Quad-Core Intel Core i5 CPU. All results in this chapter took approximately a day to compute, this is including robustness verification with combinatorial optimization solvers. Particularly robust relabeling criterion trees and robustness verification of tree ensembles for the wine dataset require much runtime. Without robust relabeling criterion trees and wine robustness verification the runtime is approximately 2 hours.

[2]`https://github.com/tudelft-cda-lab/robust-relabeling`

Table 6.2: Mean **adversarial accuracy** scores of decision trees of depth 5 on 5-fold cross validation. GROOT trees with robust relabeling and relabeling criterion trees score best against adversarial attacks. However, GROOT with relabeling runs orders of magnitude faster.

dataset	tree	tree relabeled	GROOT	GROOT relabeled	relabeling criterion
Banknote	.734 ± .077	.823 ± .035	.794 ± .049	**.824** ± .038	.811 ± .049
Breast-cancer	.874 ± .013	.903 ± .025	.912 ± .035	.922 ± .012	**.925** ± .013
Breast-cancer-d	.617 ± .158	.810 ± .026	.835 ± .013	.847 ± .014	**.851** ± .038
Sonar	.482 ± .140	.573 ± .073	.601 ± .048	**.606** ± .048	.582 ± .070
Ionosphere	.689 ± .071	.792 ± .045	.892 ± .030	.889 ± .028	**.895** ± .028
Parkinsons	.513 ± .139	.759 ± .126	.749 ± .071	.790 ± .058	**.795** ± .075
Diabetes	.708 ± .009	**.712** ± .025	.677 ± .053	**.712** ± .025	.710 ± .032
Qsar-bio.	.292 ± .060	.661 ± .004	.704 ± .029	**.736** ± .050	.686 ± .014
Spectf-heart	**.840** ± .041	**.840** ± .041	.831 ± .044	.831 ± .044	.768 ± .016
Wine	.526 ± .027	.610 ± .047	**.618** ± .043	**.618** ± .052	timeout

Table 6.3: Mean **adversarial accuracy** scores of decision tree ensembles on 5-fold cross validation. GROOT trees with robust relabeling score best against adversarial attacks, relabeled regular trees perform on average similarly to robust GROOT trees that did not use relabeling.

dataset	boosting	forest	GROOT forest	boosting relabeled	forest relabeled	GROOT forest rel.
Banknote	.786 ± .072	.846 ± .032	.851 ± .037	.822 ± .052	.849 ± .039	**.862** ± .039
Breast-cancer	.873 ± .027	.908 ± .020	.946 ± .017	.937 ± .020	.930 ± .011	**.952** ± .017
Breast-cancer-d	.606 ± .070	.745 ± .035	.805 ± .025	.821 ± .019	.842 ± .022	**.847** ± .021
Sonar	.438 ± .089	.389 ± .051	.510 ± .069	**.616** ± .076	.582 ± .034	.577 ± .048
Ionosphere	.635 ± .163	.812 ± .037	.903 ± .018	.815 ± .010	.872 ± .017	**.912** ± .021
Parkinsons	.492 ± .177	.508 ± .170	.728 ± .092	.759 ± .126	.749 ± .021	**.826** ± .066
Diabetes	.596 ± .043	.668 ± .049	.703 ± .052	.697 ± .032	.729 ± .036	**.730** ± .047
Qsar-bio.	.078 ± .025	.173 ± .015	.648 ± .046	.663 ± .002	.663 ± .002	**.781** ± .026
Spectf-heart	.863 ± .034	.891 ± .028	.888 ± .023	.877 ± .037	**.897** ± .025	.894 ± .029
Wine	.202 ± .044	.184 ± .050	.384 ± .051	.494 ± .081	.482 ± .090	**.606** ± .038

Table 6.4: Comparison of **adversarial accuracy** scores for adversarial pruning [15] and robust relabeling (ours). Adversarial pruning does not take into account that the learner can select non-robust splits where relabeling effectively removes such splits thus producing more robust models.

	Decision tree		Gradient boosting		Random forest	
dataset	Pruning	Relabeling	Pruning	Relabeling	Pruning	Relabeling
Banknote	.718 ± .060	**.823** ± .035	.809 ± .053	**.822** ± .052	**.855** ± .032	.849 ± .039
Breast-cancer	.867 ± .016	**.903** ± .025	.868 ± .031	**.937** ± .020	.906 ± .016	**.930** ± .011
Breast-cancer-d	.617 ± .158	**.810** ± .026	.619 ± .081	**.821** ± .019	.749 ± .035	**.842** ± .022
Sonar	.482 ± .140	**.573** ± .073	.438 ± .089	**.616** ± .076	.389 ± .051	**.582** ± .034
Ionosphere	.689 ± .071	**.792** ± .045	.635 ± .163	**.815** ± .010	.812 ± .037	**.872** ± .017
Parkinsons	.513 ± .139	**.759** ± .126	.492 ± .177	**.759** ± .126	.508 ± .170	**.749** ± .021
Diabetes	.708 ± .009	**.712** ± .025	.596 ± .043	**.697** ± .032	.668 ± .049	**.729** ± .036
Qsar-bio.	.262 ± .052	**.661** ± .004	.149 ± .024	**.663** ± .002	.183 ± .010	**.663** ± .002
Spectf-heart	**.840** ± .041	**.840** ± .041	.863 ± .034	**.877** ± .037	.891 ± .028	**.897** ± .025
Wine	.562 ± .030	**.610** ± .047	.212 ± .094	**.494** ± .081	.240 ± .047	**.482** ± .090

6.4 Improving Robustness

To investigate the effect of robust relabeling on adversarial robustness, we compare performance on 10 datasets with a fixed perturbation radius for each dataset. We used datasets from the UCI Machine Learning Repository [18] retrieved through OpenML [23]. All datasets, their properties and perturbation radii ϵ are listed in Table 6.1. We pre-process each dataset by scaling the features to the range $[0, 1]$. This way, we can interpret ϵ as representing a fraction of each feature's range. We compare robust relabeling to regular decision trees and ensembles trained with Scikit-learn [24], robust decision trees trained with GROOT [12] and adversarial pruning [15]. All adversarial accuracy scores were computed with optimal adversarial attacks using the GROOT toolbox[3]. For single trees, computing optimal adversarial attacks is done by enumerating all the leaves and for tree ensembles by solving the Mixed-Integer Linear Programming formulation by Kantchelian et al. [8] using GUROBI 9.1 [25].

6.4.1 Decision Trees

Decision trees have the desirable property that they are interpretable when constrained to be small enough. What exactly is the number of leaves that allow a decision tree to be interpretable is not well defined. In this work, we decide to train single trees up to a depth of 5 which enforces a maximum number of leaves of $2^5 = 32$. In Table 6.2 we compare the performance of regular, robust GROOT [12] trees and relabeling criterion trees defined in Section 6.3.1. We score the regular and GROOT trees before and after relabeling but we skip this step for relabeling criterion trees as this does not affect the learned tree.

Robust relabeling improves the performance of regular and GROOT trees significantly on most datasets and never reduces the mean adversarial accuracy. Relabeling criterion trees and relabeled GROOT trees performed similarly on average but relabeled GROOT trees run orders of magnitude faster.

6.4.2 Decision Tree Ensembles

For tasks that do not require model interpretability, it is a popular choice to ensemble multiple decision trees to create stronger models. We experiment with the robust relabeling of random forests, GROOT random forests and gradient boosting ensembles, all limited to 100 decision trees. For the gradient boosting ensembles we limit the trees to a depth of 5 to prevent overfitting. This is not required for random forests where one purposefully ensembles low bias, high variance models [17], i.e., unconstrained decision trees. We did not compare to random forests trained with the robust relabeling criterion as this was computationally infeasible. The adversarial accuracy scores before and after robust relabeling are presented in Table 6.3. On the Wine dataset we only used 100 test samples to limit the runtime.

Robust relabeling increases the mean adversarial accuracy over 5-fold cross-validation on all datasets and models. On average, the GROOT random forests with robust relabeling performed best. Clearly, the combination of robust splits and robust labeling is better than regular splits and robust labeling. Additionally we find that relabeled GROOT forests (Table 6.3) outperform relabeled GROOT trees (Table 6.2) on many datasets. This is in

[3]`https://github.com/tudelft-cda-lab/GROOT`

Table 6.5: Comparison of **regular accuracy** scores before and after applying robust relabeling. Since robustness is generally at odds with accuracy robust relabeling loses out on accuracy in about 2 out of 3 cases. However, in some cases robustness actually improves accuracy as a type of regularization.

	Decision tree		Gradient boosting		Random forest	
dataset	Before	After	Before	After	Before	After
Banknote	**.967** ± .022	.948 ± .036	**.991** ± .007	.941 ± .038	**.994** ± .005	.956 ± .020
Breast-cancer	**.969** ± .014	.958 ± .019	.962 ± .008	**.965** ± .009	.968 ± .008	**.969** ± .003
Breast-cancer-d	**.930** ± .034	.912 ± .032	.917 ± .036	**.931** ± .024	**.954** ± .017	.947 ± .028
Sonar	**.740** ± .058	.716 ± .054	.731 ± .044	**.740** ± .046	**.803** ± .031	.755 ± .054
Ionosphere	**.883** ± .034	.857 ± .047	**.906** ± .041	.863 ± .022	.926 ± .026	**.932** ± .033
Parkinsons	**.872** ± .065	.851 ± .071	**.867** ± .071	.851 ± .071	**.913** ± .082	.759 ± .014
Diabetes	.737 ± .026	**.738** ± .034	**.742** ± .026	.719 ± .028	**.769** ± .039	.768 ± .040
Qsar-bio.	**.819** ± .042	.661 ± .004	**.872** ± .023	.663 ± .002	**.868** ± .023	.663 ± .002
Spectf-heart	**.840** ± .041	**.840** ± .041	.871 ± .028	**.880** ± .033	**.897** ± .025	**.897** ± .025
Wine	**.696** ± .048	.686 ± .047	**.774** ± .024	.494 ± .081	**.786** ± .019	.498 ± .080

contrast with the original GROOT paper [12]. In that paper, large values were used for ϵ that did not allow for the models to achieve significant improvements over predicting the majority class.

6.4.3 Adversarial Pruning

Adversarial pruning [15] is a technique that implicitly prunes models by removing samples from the training dataset that are not well separated ($D \setminus C$). This intuitively makes models more robust as the models then explicitly ignore samples that will make the models more susceptible to adversarial attacks. Using decision tree learning algorithms as-is on this dataset ($D \setminus C$), without taking robustness into account, still provides models that suffer from adversarial examples. In Table 6.4 we compare the adversarial robustness of models trained with adversarial pruning and robust relabeling. We notice that on many datasets, adversarial pruning only removes a small number of samples which results in models that are similar to the fragile models produced by regular decision tree learning algorithms.

6.4.4 Accuracy Robustness Trade-off

Since we optimize robustness by enforcing samples to be correctly classified in a region around each sample, there can be a cost in regular accuracy. In Table 6.5 we compare the accuracy of regular models with and without robust relabeling. We find that indeed robust relabeling reduces regular accuracy in approximately two out of three cases that we tested. However, there are also instances where accuracy actually improves, such as in the case of gradient boosting on the breast cancer datasets. We expect that robustification has a helpful regularization effect in these situations.

6.5 Regularizing Decision Trees

Robust relabeling regularizes decision trees and tree ensembles by changing the leaf labels to maximize adversarial robustness. To understand the regularization effect we first visualize models before and after robust relabeling on toy datasets. Additionally, we contrast the regularization effect of robust relabeling with Cost Complexity Pruning. We show that while both methods can improve test accuracy, robust relabeling favors robustness while Cost Complexity Pruning favors regular accuracy.

6.5.1 Toy Datasets

To understand the effects of robust relabeling we generate three two-dimensional datasets and visualize the decision regions before and after relabeling. In Figure 6.3 we train decision trees, random forests and gradient boosting with 5% label noise and adversarial attacks within an L^{∞} radius of $\epsilon = 0.05$. All features are scaled to the range $[0, 1]$ therefore ϵ represents 5% of each feature's range. The boosted and single decision trees were limited to a depth of 5.

In all types of models we see that there are small regions with a wrong prediction in areas where the model predicts the correct label. For instance, in the normal decision tree trained on 'moons', we see a slim orange leaf in the region that is otherwise predicted as blue. This severely reduces the robustness of the model against adversarial attacks since nearby samples can be perturbed into those regions. Robust relabeling effectively removes these leaves from the models to improve adversarial robustness. Although the effect of regularization of decision trees is hard to quantify, we intuitively see that the relabeled models are more consistent in their predictions. We expect this property to also improve the explainability of the models by methods such as counterfactual explanations [26].

6.5.2 Comparison with Cost Complexity Pruning

Cost Complexity Pruning is a method that reduces the size of decision trees to improve their generalization capabilities. This method iteratively merges leaves that have a lower increase in predictive performance than some user-defined threshold α. In Figure 6.4 we compare the effects of Cost Complexity Pruning and robust relabeling on the Diabetes dataset. Here, we trained decision trees without size constraints and varied the hyperparameters α and ϵ then measured accuracy before and after adversarial attacks.

While the effectiveness of cost complexity pruning and robust relabeling varies between datasets we find that generally both methods can increase test accuracy compared to the baseline model. However, cost complexity pruning achieved better accuracy scores on average while robust relabeling achieved better adversarial accuracy scores. Clearly, there is a difference between regularization for generalization and adversarial robustness. Such a trade-off between accuracy and robustness has been widely described in the literature [27, 28].

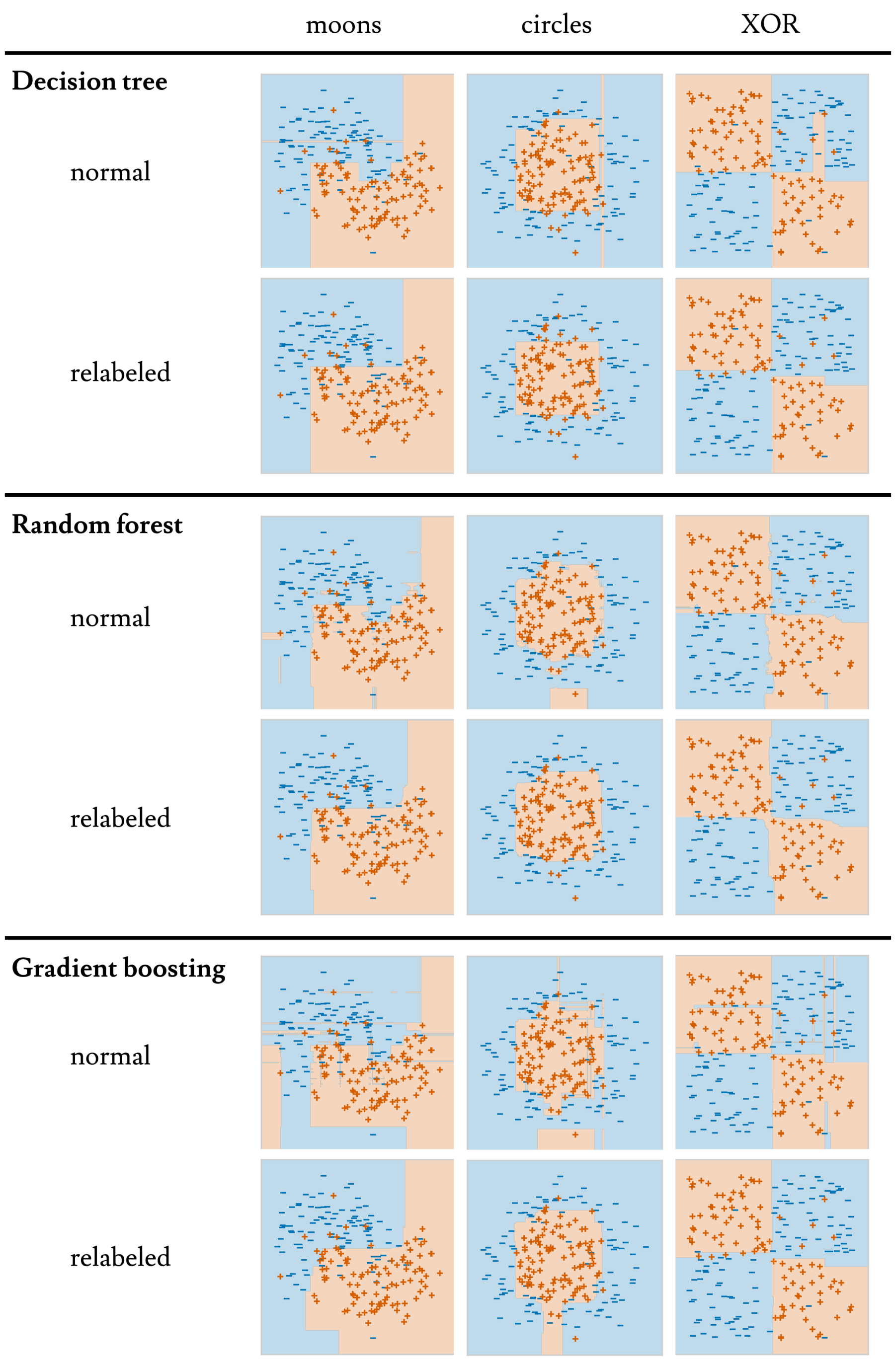
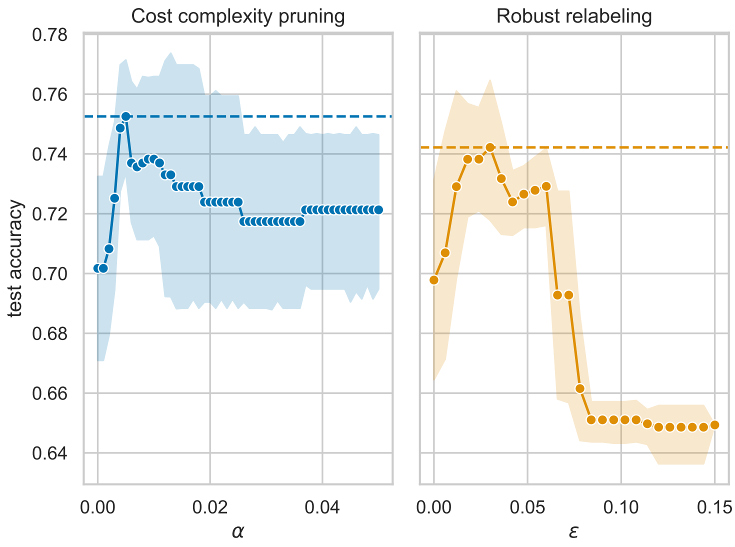
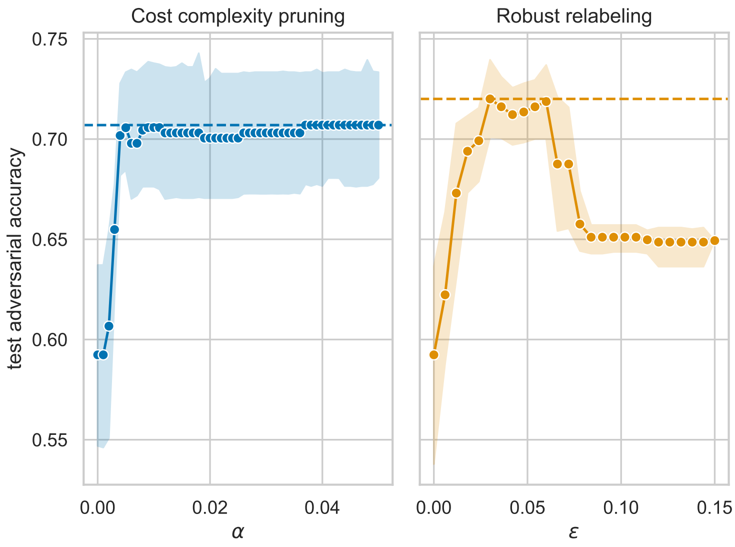


Figure 6.3: Decision regions of tree models before and after robust relabeling. Robust relabeling effectively removes fragile regions resulting in visually simpler models.



(a) Test accuracy

6



(b) Test accuracy against $\epsilon = 0.01$ attacks

Figure 6.4: Test scores on the Diabetes dataset when varying the hyperparameters of cost complexity pruning and robust relabeling. Both improve upon unpruned trees ($\alpha = \epsilon = 0$), but cost complexity pruning performs better at regular accuracy while robust relabeling enhances adversarial accuracy.

6.6 Conclusions

In this work, we studied relabeling as a method to improve the adversarial robustness of decision trees and their ensembles. As training optimal robust decision trees is expensive and training heuristic robust trees inexact, we propose a polynomial-time post-learning algorithm to overcome these problems: robust relabeling. Our results show that robust relabeling significantly improves the robustness of regular and robust tree models. Robustly relabeling models trained with the state-of-the-art robust tree heuristic GROOT further improved the performance. While we can also use the robust relabeling method during the tree learning procedure this took up to hours of runtime and produced decision trees that were approximately as robust as relabeled GROOT trees.

We expect robust relabeling in combination with methods such as GROOT to become important for training models that get deployed in adversarial contexts such as fraud or malware detection. The result that finding an optimal robust labeling can be done in polynomial time can help to further improve methods for optimal robust decision tree learning. In future work, we will explore the regularity effects of robust models for instance for improved counterfactual explanations.

Bibliography

[1] D. Vos and S. Verwer, *Adversarially robust decision tree relabeling,* in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (Springer, 2022) pp. 203–218.

[2] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,* Nature Machine Intelligence **1**, 206 (2019).

[3] L. Breiman, *Random forests,* Machine learning **45**, 5 (2001).

[4] J. H. Friedman, *Greedy function approximation: a gradient boosting machine,* Annals of statistics , 1189 (2001).

[5] T. Chen and C. Guestrin, *Xgboost: A scalable tree boosting system,* in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[6] A. V. Dorogush, V. Ershov, and A. Gulin, *Catboost: gradient boosting with categorical features support,* arXiv preprint arXiv:1810.11363 (2018).

[7] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, *Lightgbm: A highly efficient gradient boosting decision tree,* Advances in neural information processing systems **30** (2017).

[8] A. Kantchelian, J. D. Tygar, and A. Joseph, *Evasion and hardening of tree ensemble classifiers,* in *International conference on machine learning* (PMLR, 2016) pp. 2387–2396.

[9] C. Zhang, H. Zhang, and C.-J. Hsieh, *An efficient adversarial attack for tree ensembles,* Advances in Neural Information Processing Systems **33**, 16165 (2020).

[10] H. Chen, H. Zhang, D. Boning, and C.-J. Hsieh, *Robust decision trees against adversarial examples,* in *International Conference on Machine Learning* (PMLR, 2019) pp. 1122–1131.

[11] S. Calzavara, C. Lucchese, G. Tolomei, S. A. Abebe, and S. Orlando, *Treant: training evasion-aware decision trees,* Data Mining and Knowledge Discovery **34**, 1390 (2020).

[12] D. Vos and S. Verwer, *Efficient training of robust decision trees against adversarial examples,* in *International Conference on Machine Learning* (PMLR, 2021) pp. 10586–10595.

[13] D. Vos and S. Verwer, *Robust optimal classification trees against adversarial examples,* arXiv preprint arXiv:2109.03857 (2021).

[14] N. Justin, S. Aghaei, A. Gomez, and P. Vayanos, *Optimal robust classification trees,* in *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond* (2021).

[15] Y.-Y. Yang, C. Rashtchian, Y. Wang, and K. Chaudhuri, *Robustness for non-parametric classification: A generic attack and defense,* in *International Conference on Artificial Intelligence and Statistics* (PMLR, 2020) pp. 941–951.

[16] Y. Wang, S. Jha, and K. Chaudhuri, *Analyzing the robustness of nearest neighbors to adversarial examples,* in *International Conference on Machine Learning* (PMLR, 2018) pp. 5133–5142.

[17] L. Breiman, *Bagging predictors,* Machine learning **24**, 123 (1996).

[18] D. Dua and C. Graff, *UCI machine learning repository,* (2017).

[19] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees* (Taylor & Francis, 1984).

[20] F. Kamiran, T. Calders, and M. Pechenizkiy, *Discrimination aware decision tree learning,* in *2010 IEEE International Conference on Data Mining* (IEEE, 2010) pp. 869–874.

[21] M. Velikova and H. Daniels, *Decision trees for monotone price models,* Computational Management Science **1**, 231 (2004).

[22] J. E. Hopcroft and R. M. Karp, *An n^5/2 algorithm for maximum matchings in bipartite graphs,* SIAM Journal on computing **2**, 225 (1973).

[23] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, *Openml: Networked science in machine learning,* SIGKDD Explorations **15**, 49 (2013).

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, *Scikit-learn: Machine learning in python,* the Journal of machine Learning research **12**, 2825 (2011).

[25] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual,* (2022).

[26] S. Verma, J. Dickerson, and K. Hines, *Counterfactual explanations for machine learning: A review,* arXiv preprint arXiv:2010.10596 (2020).

[27] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry, *Robustness may be at odds with accuracy,* arXiv preprint arXiv:1805.12152 (2018).

[28] H. Zhang, Y. Yu, J. Jiao, E. Xing, L. El Ghaoui, and M. Jordan, *Theoretically principled trade-off between robustness and accuracy,* in *International conference on machine learning* (PMLR, 2019) pp. 7472–7482.

7 Poisoning Robust and Differentially-Private Decision Trees

Decision trees are interpretable models that are well-suited to non-linear learning problems. Much work has been done on extending decision tree learning algorithms with differential privacy, a system that guarantees the privacy of samples within the training data. However, current state-of-the-art algorithms for this purpose sacrifice much utility for a small privacy benefit, which makes them less likely to be used in practice. These solutions create random decision nodes that reduce decision tree accuracy or spend an excessive share of the privacy budget on labeling leaves. Moreover, many works do not support continuous features or leak information about them. We propose a new method called PrivaTree based on histograms that chooses good splits while consuming the privacy budget efficiently. The resulting trees provide a better privacy-utility trade-off and accept mixed numerical and categorical data without leaking information about numerical features. Finally, while it is notoriously hard to give robustness guarantees against data poisoning attacks, we demonstrate bounds for the expected accuracy and success rates of backdoor attacks against differentially-private learners. By leveraging the better privacy-utility trade-off of PrivaTree we can train decision trees with significantly better robustness against backdoor attacks compared to regular decision trees and with meaningful theoretical guarantees. In this way, PrivaTree provides a secondary incentive for adoption.

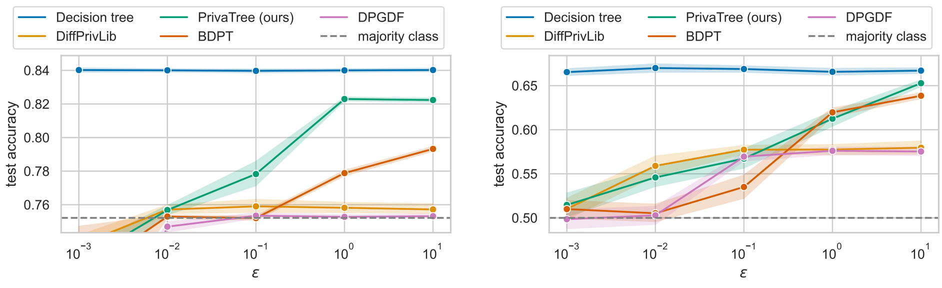
7.1 Introduction

Machine learning has achieved widespread success with neural networks and ensemble methods, but it is almost impossible for humans to understand the decisions such models make [1]. Fortunately, much work has been done on training machine learning models that are directly interpretable by humans [2]. Especially size-limited decision trees [3, 4] are successful methods for their interpretability combined with their ability to predict non-linear data.

While decision trees can offer interpretability, they reveal information about the data they were trained on. This is a detrimental property when models are trained on private data that contains sensitive information, such as in fraud detection and medical applications. Differentially-private [5] machine learning models solve this problem by introducing carefully crafted randomness into the way the models are trained [6]. For differentially-private decision trees, the entire model consisting of decision node splits and leaf labels can be made public, and by extension, predictions made by the model. This is not only useful for training interpretable private models, but decision trees are also vital primitives for building ensembles [7–10] for tabular data. The key problem in training differentially-private models is efficiently spending the privacy budget ϵ to achieve high utility. In this work, we propose an algorithm for training decision trees with an improved privacy-utility trade-off.

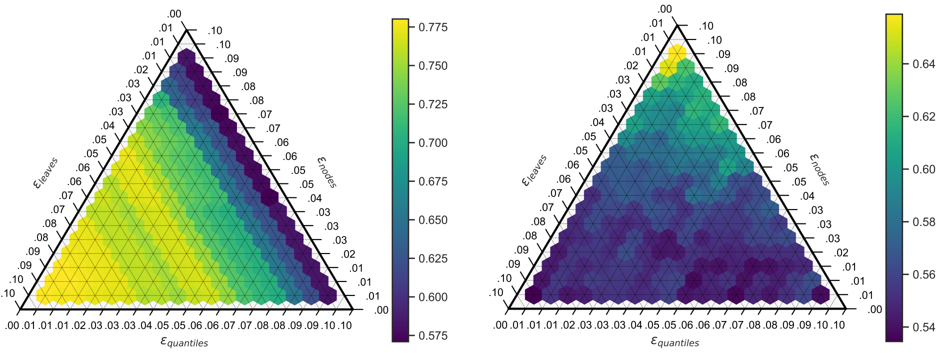
Many previous works have already proposed ways to generate differentially-private decision trees, but they also have shortcomings. There are two main categories of algorithms here. The first category [11–14] chooses splits completely at random and allocates the entire privacy budget for labeling the leaves. The second category [15–18] extends the greedy splitting procedure of regular decision trees, where splits are selected by optimizing a splitting criterion. These works guarantee differential privacy by incorporating noise into the splitting criterion while consuming a part of the user-defined privacy budget. However, since these approaches require computing many scores to select good decision nodes, an inefficient use of a mechanism results in consuming a large fraction of the privacy budget. The remaining budget is spent on labeling leaves.

In this work, we propose a method called PrivaTree to train differentially-private decision trees. PrivaTree uses the privacy budget more efficiently than previous work when choosing splits by leveraging histograms and using the permute-and-flip mechanism. We also propose a strategy for distributing the privacy budget that offers good performance for both very small and large datasets. The result is a practical method for training private trees with a significantly better utility for all privacy budgets. PrivaTree also prevents leakage from the location of splits on numerical features, a property that some previous methods do not have. Moreover, we demonstrate how the ability to train accurate differentially-private decision trees with small privacy budgets allows for performance guarantees against data poisoning attacks in which an adversary manipulates the training data. Our experiments on various tabular benchmark datasets demonstrate an improved privacy-utility trade-off compared to other private trees. We also experimentally demonstrate that PrivaTree offers stronger poisoning robustness guarantees than private logistic regression [19], another interpretable method. Our experiments on the MNIST 0 vs 1 dataset show that, indeed, PrivaTrees with small privacy budgets resist a trigger-based backdoor attack.



(a) *adult* dataset (45222 samples, 14 features) (b) *compas-two-years* (4966 samples, 11 features)

Figure 7.1: Mean accuracy scores of depth 4 trees when varying privacy budget ϵ from private to less private with 50 repetitions. For extremely small privacy budgets, DiffPrivLib performs best, but for higher budgets, PrivaTree achieves a significantly higher accuracy.



(a) *vote* dataset (435 samples, 16 features) (b) *pol* dataset (10082 samples, 26 features)

Figure 7.2: Training accuracy scores when varying the distribution of the privacy budget $\epsilon = 0.1$ over different parts of the PrivaTree algorithm with depth 3. Scores were averaged over 50 executions. On small datasets such as *vote*, the best trees allocate more budget to the leaves, while large datasets such as *pol* benefit from more budget for splitting.

7.2 Improving differentially private decision trees: PrivaTree

In this section, we present PrivaTree, which is an algorithm for training differentially-private decision trees with high utility. PrivaTree incorporates three techniques to improve performance:

- Histograms to limit the number of splits to consider for each decision node.
- A better distribution of privacy budget based on a bound for labeling leaves accurately.
- The permute-and-flip mechanism instead of the exponential mechanism for leaf labeling and node splitting.

Additionally, PrivaTree pre-processes numerical features into bins which enables it to train on both numerical and categorical features. PrivaTree assumes that the range of numerical features, the set of categorical values, the set of class labels, and the dataset size are publicly known. In our experiments, we compute these values based on the complete datasets. We provide pseudocode in Algorithm 7, which we describe in more detail in the rest of this section.

7.2.1 Scoring splits using histograms

Classification tree learning algorithms such as CART create decision nodes by recursively choosing a split among all feature values that minimizes the weighted Gini impurity. Previous differentially-private decision trees used similar approaches but often leak information about numerical feature values. Friedman and Schuster [18] show how to split numerical features using the continuous exponential mechanism privately, but this requires more privacy budget because each feature needs to be considered separately. Another work, BDPT, permits some leakage, providing a weaker form of privacy by averaging every 5 numerical feature values, and then splitting using the exponential mechanism.

Numerical features To find high-quality splits while protecting feature value information and efficiently using the privacy budget, we use histograms. Splitting according to histograms has been a successful progression in decision tree learning for gradient boosting ensembles [9, 10] where it is used for its runtime efficiency. Instead, we rely on them because they reduce the number of threshold values considered for splitting which improves node selection at limited privacy budget. Specifically, we compute the Gini impurity for every threshold value between the bins of all features and use the permute-and-flip mechanism with to choose the one that privately minimizes impurity. This results in decision node rules of the kind 'feature value $\leq$ threshold'. In the rest of this paper, we fix the number of bins for numerical features to 10. The global sensitivity of this operation is $\frac{1}{2}$ as this is the range of the Gini impurity, but the performance can be improved using smooth sensitivity [20]. Therefore we first privately count the number of samples that reach the node using the Geometric mechanism and budget $\frac{1}{2}\epsilon_{\text{node}}$ and then use the smooth sensitivity $1-(\frac{n}{n+1})^2-(\frac{1}{n+1})^2$ for the permute-and-flip mechanism with budget $\frac{1}{2}\epsilon_{\text{node}}$.

Categorical features Previous decision tree learning algorithms support categorical features by only splitting one category at a time. While this method is sound, it often requires deep trees to make enough splits for categorical features to be useful, which harms interpretability. In the PrivaTree algorithm, we find a partition of the categories instead. Such rules are of the kind ‘categories L go left, categories R go right’. This is done by fixing the order of the categories and finding a threshold somewhere in this sequence of categories, similar to finding a split on numerical data.

7.2.2 Pre-processing by binning

Before the splitting procedure, PrivaTree must select the boundaries of the 10 bins of the histograms for each numerical feature. A natural choice is to create equal-width bins of numerical features based on the public knowledge of each feature’s range, but this approach can have problems: features with long tails would cause data to be concentrated in a few bins and result in a large loss of information. Instead, we also consider binning numerical features using quantiles so that they evenly divide the samples over each bin. Since regular quantiles leak information about the dataset, we also implement differentially-private quantiles. We use the *jointexp* algorithm [21] for this, which improves performance over optimal statistical estimation techniques [22] when computing multiple quantiles on the same data. This algorithm runs with privacy budget $\epsilon_{\text{quantile}}$, divided over each feature using the sequential composition property we will define later. Categorical variables require no preprocessing. Instead, we encode them as integers, and the split-finding operation handles these values natively. In our results, we compare quantiles with equal-width bins and suggest equal-width bins as a good baseline, as computing them does not consume the privacy budget.

7.2.3 Leaf labeling according to majority votes

Once the PrivaTree algorithm has produced a series of decision nodes by recursively splitting and reaches the stopping criterion, the algorithm creates a leaf containing a prediction. In the non-private setting, the prediction is usually chosen to be the majority of the class labels of samples that reach that leaf to maximize accuracy. However, this leaks private information. Previous works have used the Laplace or (smooth) exponential mechanisms, but like modern implementations such as DiffPrivLib [13], we opt for the permute-and-flip mechanism [23]. Permute-and-flip is proven to have an expected error that is at least as good or better than that of the exponential mechanism and practically outperforms it. To label a leaf, we count the number of samples of each class and apply permute-and-flip with privacy budget ϵ_{leaf}, which we define later.

7.2.4 Distributing the privacy budget

The composability property of differential privacy allows modular algorithm design by breaking up the algorithm into differentially-private primitives. However, it is generally not obvious how to distribute the privacy budget ϵ over the primitives to maximize the expected utility of the outcomes. Previous works have tried various heuristics, such as distributing ϵ equally over each private operation or distributing epsilon 50-50 between node and leaf operations. To understand the role of budget distribution in private tree learning, we visualized the average training accuracy over 50 runs when varying budgets

Algorithm 7 Train PrivaTree with ϵ differential privacy

Input: dataset X $(n \times m)$, labels y, privacy budget ϵ, maximum leaf error $\mathcal{E}_{max}$, maximum depth d

```
 1: $\epsilon_{\text{leaf}} = \min(\frac{\epsilon}{2}, \epsilon'_{\text{leaf}})$, where $\epsilon'_{\text{leaf}}$ is computed with Equation 7.1
 2: $\epsilon_{\text{node}} = (\epsilon - \epsilon_{\text{leaf}}) \cdot \frac{1}{d}$
 3: for numerical features $j_{\text{num}}$ do
 4:     bin values $X_{i,j_{\text{num}}}$ into bins $B_{j_{\text{num}}}$ of equal width (determined with public feature range)
 5: end for
 6: procedure FitTree$(X', y', d')$
 7:     if $d' = 0$ then
 8:         compute class counts $N_0, N_1, ..., N_K$ for all $K$ classes
 9:         return Leaf$(\mathcal{M}_{PF}(\langle N_0, N_1, ..., N_K \rangle))$    ▷ with budget $\epsilon_{\text{leaf}}$ and sensitivity 1
10:     else
11:         create histograms $\forall j, k, b \in B_j : H_{j,b,k} \leftarrow \sum_i [X'_{i,j}{=}b \wedge y'_i{=}k]$
12:         compute utilities with the negative Gini impurity $u_{j,b} \leftarrow -\text{Gini}(H_{j,b,0}, H_{j,b,1}, ...)$
13:         count the number of samples that reach this node $n = \mathcal{M}_{Geom}(|X'|)$    ▷ with budget $\frac{1}{2}\epsilon_{\text{node}}$ and sensitivity 1
14:         choose split $(j^*, b^*)$ with $\mathcal{M}_{PF}(\{u_{j,b} : \forall j, k\})$    ▷ with budget $\frac{1}{2}\epsilon_{\text{node}}$ and sensitivity $1-(\frac{n}{n+1})^2-(\frac{1}{n+1})^2$
15:         partition $X'$ into $(X^{\text{left}}, X^{\text{right}})$, and $y'$ into $(y^{\text{left}}, y^{\text{right}})$ according to $(j^*, b^*)$
16:         $\mathcal{T}_{\text{left}} \leftarrow$ FitTree$(X^{\text{left}}, y^{\text{left}}, d'-1)$,    $\mathcal{T}_{\text{right}} \leftarrow$ FitTree$(X^{\text{right}}, y^{\text{right}}, d'-1)$
17:         return Node$(j^*, b^*, \mathcal{T}_{\text{left}}, \mathcal{T}_{\text{right}})$
18:     end if
19: end procedure
20: return FitTree$(X, y, d)$
```

were spent on parts of the algorithm (quantile finding, node splitting, and leaf labeling) in Figure 7.2. We find that when ϵ is large compared to the dataset size, it is best to spend nearly all the budget on node operations. When ϵ is small compared to the dataset size, spending the budget on leaf labeling is favorable, i.e., creating nearly random nodes and accurately labeling the leaves. Computing quantiles usually requires too much budget to be useful, so by default, we decide to split features into equal-width bins according to the feature range (which is assumed to be public knowledge). For example, if 'age' is a feature with values between 0 and 120 years, splitting it into 10 bins results in bins 0-12, 12-24, and 106-120. By noticing that we can bound the expected error incurred by labeling leaves for a given privacy budget, we propose a budget distribution scheme that scales well for varying values of ϵ. When the privacy budget is low compared to the dataset size, set $\epsilon_{\text{leaf}} = \frac{\epsilon}{2}$, i.e. half of the budget. When the budget is relatively high, set ϵ_{leaf} such that the maximum expected error incurred by labeling $\mathbb{E}[\mathcal{E}(\mathcal{M}, \vec{N})]$ is at most equal to the user-specified labeling error limit $\mathcal{E}_{max}$. Distribute the remaining budget $\epsilon - \epsilon_{\text{leaf}}$ over node operations to improve algorithm utility for higher values of ϵ. For all our results, we used $\mathcal{E}_{max} = 0.01$ i.e., if possible, we only allow an extra accuracy loss of 1% due to leaf labeling errors.

Corollary 1. *For K classes, n samples and depth d trees, the amount of privacy budget ϵ'_{leaf} needed for labeling leaves with the permute-and-flip mechanism with expected error $\mathbb{E}[\mathcal{E}(\mathcal{M}_{PF},\vec{N})]$ of at most $\mathcal{E}_{max}$ is:*

$$\epsilon'_{\text{leaf}} \leq \frac{2^d \max_p 2\log(\frac{1}{p})\left(1 - \frac{1-(1-p)^K}{Kp}\right)}{n\,\mathcal{E}_{max}}, \tag{7.1}$$

Proof. This result naturally follows from applying the worst-case error bound proved for $\mathcal{M}_{PF}$ in the permute-and-flip paper [23]. A complete proof is given in the appendix[1]. □

Following the definition of ϵ'_{leaf} this results in the privacy budget being distributed as

$$\epsilon_{\text{leaf}} = \min\left(\frac{\epsilon}{2}, \epsilon'_{\text{leaf}}\right), \tag{7.2}$$

$$\epsilon_{\text{node}} = \frac{\epsilon - \epsilon_{\text{leaf}}}{d}. \tag{7.3}$$

Next, we show that this budget distribution scheme actually provides differential privacy by showing that the composition of mechanisms spends privacy budget equal to ϵ.

Theorem 7. *PrivaTree, as described in Algorithm 7, provides ϵ-differential privacy.*

Proof. For numerical attributes in the training set X, PrivaTree first computes bins using public knowledge, which does not consume privacy budget. After that, the algorithm recursively splits the root node, and since each split creates two distinct partitions of the data, each data point is only used in d nodes where d is the maximum depth of the tree. The amount of budget spent on node splitting is then $d \cdot \epsilon_{\text{node}}$ through sequential composition. Finally, leaf labeling consumes ϵ_{leaf} of the privacy budget, and since each data point is only used in a single leaf, this follows parallel composition (spending ϵ_{leaf} once). Combining all the operations, we find that the amount of privacy budget spent is:

$$\begin{aligned}\epsilon &= d \cdot \epsilon_{\text{node}} + \epsilon_{\text{leaf}} \\ &= d \cdot \frac{\epsilon - \epsilon_{\text{leaf}}}{d} + \epsilon_{\text{leaf}} \\ &= \epsilon - \epsilon_{\text{leaf}} + \epsilon_{\text{leaf}} \\ &= \epsilon.\end{aligned}$$

Therefore Algorithm 7 provides ϵ-differential privacy. □

7.3 Poisoning Robustness

When using machine learning trained on crowd-sourced data, such as in federated learning scenarios, or on potentially manipulated data one has to consider malicious user behavior. One such threat is data poisoning, in which users insert x data points into the training dataset to confuse the classifier or introduce a backdoor. Many defenses have been proposed, such as using learning behavior to ignore backdoor data [24] or post-processing

[1] `https://github.com/tudelft-cda-lab/PrivaTree`

based on adversarial robustness to remove backdoors [25], but such methods work heuristically and offer no guarantees. We use the fact that ϵ-differentially-private machine learning algorithms are limited in sensitivity to dataset changes to provide guarantees against poisoning attacks. Ma et al. [26] introduce the attack cost $J(D) = \mathbb{E}[C(\mathcal{M}(D))]$, i.e. the expected value of a cost function C that an attacker gets from models produced by $\mathcal{M}$ on dataset D. They prove the following about the cost of attacks against differentially-private learners:

Theorem 8. *[26] Let $\mathcal{M}$ be an ϵ-differentially-private learner. Let $J(\tilde{D})$ be the attack cost, where $\tilde{D} \ominus D \leq x$ (i.e. a dataset with x poisoned samples compared to clean dataset D), then*

$$J(\tilde{D}) \geq e^{-x\epsilon} J(D), \quad (C \geq 0) \tag{7.4}$$

$$J(\tilde{D}) \geq e^{x\epsilon} J(D). \quad (C \leq 0) \tag{7.5}$$

We will consider two scenarios with different attacker objectives and their associated cost functions:

- An attacker adds or removes up to x samples from a dataset in an attempt to maximally reduce the accuracy of a model learned from the dataset. The associated cost function is the accuracy of the model, i.e. $J_{\text{Acc}}(\tilde{D}) = \mathbb{E}[\text{Accuracy}(\mathcal{M}(\tilde{D}))]$.
- An attacker adds x copies of data points with a trigger pattern inserted into them and the associated label is changed to a target label in an effort to create a backdoor. The associated cost function is the Attack Success Rate (ASR), i.e. the percentage of data points for which the predicted label can be flipped to the target label. Since Theorem 8 assumes (without loss of generality) that the attacker minimizes J, we use the complement of the ASR as the cost, as the attacker wants to maximize ASR: $J_{\text{ASR}}(\tilde{D}) = \mathbb{E}[1 - \text{ASR}(\mathcal{M}(\tilde{D}))]$.

These two settings lead to the following robustness guarantees that we empirically evaluate in Section 4.5 for various private learners and datasets.

Corollary 2. *Let $\mathcal{M}$ be an ϵ-differentially-private learner. Let $J_{Acc}(\tilde{D}) = \mathbb{E}[Accuracy(\mathcal{M}(\tilde{D}))]$ be the model's expected accuracy on the poisoned dataset $\tilde{D}$, where $\tilde{D} \in B(D, x)$ (i.e. a dataset with x poisoned samples), then*

$$\mathbb{E}[Accuracy(\mathcal{M}(\tilde{D}))] \geq e^{-x\epsilon}\ \mathbb{E}[Accuracy(\mathcal{M}(D))]$$

Proof. Since the accuracy score function is non-negative this follows directly from Equation 7.4:

$$J(\tilde{D}) \geq e^{-x\epsilon} J(D)$$
$$\mathbb{E}[\text{Accuracy}(\mathcal{M}(\tilde{D}))] \geq e^{-x\epsilon}\ \mathbb{E}[\text{Accuracy}(\mathcal{M}(D))]\,. \qquad \square$$

Corollary 3. *Let $\mathcal{M}$ be an ϵ-differentially-private learner. Let $J_{ASR}(\tilde{D}) = \mathbb{E}[1 - ASR(\mathcal{M}(\tilde{D}))]$ be the expected backdoor attack success rate on the poisoned dataset $\tilde{D}$, where $\tilde{D} \in B(D, x)$ (i.e. a dataset with x poisoned samples), then*

$$\mathbb{E}[ASR(\mathcal{M}(\tilde{D}))] \leq 1 - e^{-x\epsilon}\ \mathbb{E}[1 - ASR(\mathcal{M}(D))]$$

Proof. Since J_{ASR} is non-negative we derive the guarantee from Equation 7.4:

$$\begin{aligned}
J(\tilde{D}) &\geq e^{-x\epsilon} J(D) \\
\mathbb{E}[1 - \text{ASR}(\mathcal{M}(\tilde{D}))] &\geq e^{-x\epsilon}\ \mathbb{E}[1 - \text{ASR}(\mathcal{M}(D))] \\
\mathbb{E}[\text{ASR}(\mathcal{M}(\tilde{D}))] - \mathbb{E}[1] &\leq -e^{-x\epsilon}\ \mathbb{E}[1 - \text{ASR}(\mathcal{M}(D))] \\
\mathbb{E}[\text{ASR}(\mathcal{M}(\tilde{D}))] &\leq 1 - e^{-x\epsilon}\ \mathbb{E}[1 - \text{ASR}(\mathcal{M}(D))]\ .
\end{aligned}$$

□

OpenML dataset	**decision tree**	**BDPT**	**PrivaTree***	**DPGDF**	**DiffPrivLib**	**PrivaTree**
	no privacy	leaking numerical splits		differential privacy		
			Numerical data			
Bioresponse	.711 ± .006	.508 ± .009	**.553** ± .021	-	**.517** ± .008	.505 ± .003
Diabetes130US	.606 ± .001	.515 ± .006	**.556** ± .010	-	.522 ± .009	**.529** ± .011
Higgs	.657 ± .001	timeout	**.646** ± .002	-	.514 ± .004	**.593** ± .003
MagicTelescope	.781 ± .006	.500 ± .000	**.687** ± .025	-	**.663** ± .038	.658 ± .025
MiniBooNE	.871 ± .001	.500 ± .000	**.806** ± .008	-	.503 ± .001	**.763** ± .011
bank-marketing	.771 ± .005	.499 ± .000	**.689** ± .019	-	**.560** ± .022	.555 ± .031
california	.783 ± .002	.500 ± .000	**.716** ± .014	-	.593 ± .039	**.708** ± .031
covertype	.740 ± .001	.501 ± .001	**.740** ± .002	-	.531 ± .007	**.728** ± .001
credit	.748 ± .001	.500 ± .000	**.689** ± .025	-	**.528** ± .010	.523 ± .014
default-of-credit.	.704 ± .002	.500 ± .000	**.593** ± .015	-	.565 ± .030	**.573** ± .028
electricity	.734 ± .002	.500 ± .000	**.728** ± .004	-	**.633** ± .020	.617 ± .011
eye_movements	.574 ± .003	.500 ± .001	**.517** ± .006	-	**.516** ± .006	.510 ± .005
heloc	.704 ± .004	.526 ± .009	**.605** ± .023	-	.559 ± .020	**.587** ± .027
house_16H	.815 ± .004	.500 ± .000	**.690** ± .019	-	.533 ± .007	**.593** ± .032
jannis	.715 ± .002	.500 ± .000	**.651** ± .007	-	.530 ± .005	**.633** ± .012
pol	.929 ± .003	.538 ± .023	**.663** ± .048	-	.569 ± .032	**.578** ± .042
			Numerical & categorical data			
albert	.640 ± .002	.500 ± .000	**.624** ± .003	.511 ± .005	.532 ± .018	**.533** ± .016
compas-two-years	.672 ± .006	.576 ± .021	**.617** ± .016	.568 ± .006	**.585** ± .012	.574 ± .012
covertype	.756 ± .000	.606 ± .007	**.745** ± .002	.535 ± .008	.545 ± .022	**.744** ± .002
default-of-credit.	.707 ± .004	.500 ± .000	**.562** ± .013	.528 ± .005	**.584** ± .026	.572 ± .033
electricity	.732 ± .002	.516 ± .015	**.726** ± .004	.517 ± .002	.539 ± .004	**.616** ± .006
eye_movements	.579 ± .001	.506 ± .006	**.533** ± .009	**.531** ± .007	.506 ± .004	.511 ± .012
road-safety	.728 ± .001	.685 ± .001	**.704** ± .002	.685 ± .001	.554 ± .032	**.692** ± .004
			UCI datasets (numerical & categorical)			
adult	.840 ± .001	.752 ± .000	**.776** ± .014	.751 ± .000	.758 ± .005	**.782** ± .012
breast-w	.950 ± .007	.669 ± .028	**.902** ± .013	-	**.921** ± .011	.887 ± .038
diabetes	.734 ± .006	**.622** ± .023	.611 ± .042	-	**.678** ± .018	.654 ± .008
mushroom	.971 ± .001	**.855** ± .043	.788 ± .035	.694 ± .045	.759 ± .022	**.867** ± .036
nursery	1.000 ± .000	.998 ± .002	**1.000** ± .000	.701 ± .017	.753 ± .063	**1.000** ± .000
vote	.944 ± .013	.647 ± .057	**.857** ± .042	.596 ± .098	**.854** ± .050	.749 ± .054
Total average	.762	.572	**.689**	-	.600	**.649**

Table 7.1: 5-fold cross-validated mean test accuracy and standard errors at ϵ=0.1 for trees of depth 4. PrivaTree* uses non-private quantiles, DPGDF only ran on categorical features. PrivaTree outperforms existing methods on most datasets.

				guarantee		
dataset	**method**	ϵ	**acc.**	**0.1%**	**0.5%**	**1%**
Bioresponse	PrivaTree	.01	.495	.486	.435	.378
		.1	**.515**	.421	.140	.035
	diffprivlib LR	.01	.509	**.499**	**.447**	**.389**
		.1	.501	.411	.137	.034
Diabetes130US	PrivaTree	.01	.509	**.291**	**.030**	**.002**
		.1	**.543**	.002	.000	.000
	diffprivlib LR	.01	.484	.276	.028	.002
		.1	.506	.002	.000	.000
Higgs	PrivaTree	.01	.537	.000	.000	.000
		.1	**.596**	.000	.000	.000
	diffprivlib LR	.01	.491	.000	.000	.000
		.1	**.596**	.000	.000	.000
MagicTelescope	PrivaTree	.01	.585	**.529**	**.344**	**.201**
		.1	**.654**	.241	.003	.000
	diffprivlib LR	.01	.566	.512	.333	.194
		.1	.601	.221	.003	.000
MiniBooNE	PrivaTree	.01	.502	.281	.027	.001
		.1	**.763**	.002	.000	.000
	diffprivlib LR	.01	.587	**.328**	**.032**	**.002**
		.1	.608	.002	.000	.000
bank-marketing	PrivaTree	.01	.522	.482	.343	.225
		.1	**.650**	.292	.010	.000
	diffprivlib LR	.01	.537	**.496**	**.353**	**.232**
		.1	.484	.217	.007	.000
california	PrivaTree	.01	.533	**.454**	**.235**	**.102**
		.1	**.692**	.140	.000	.000
	diffprivlib LR	.01	.517	.440	.228	.099
		.1	.434	.088	.000	.000
covertype	PrivaTree	.01	.727	**.008**	.000	.000
		.1	**.731**	.000	.000	.000
	diffprivlib LR	.01	.584	.006	.000	.000
		.1	.615	.000	.000	.000
credit	PrivaTree	.01	.508	**.446**	**.262**	**.134**
		.1	**.529**	.144	.001	.0
	diffprivlib LR	.01	.472	.415	.244	.125
		.1	.471	.128	.001	.0
default-of-credit.	PrivaTree	.01	.523	**.473**	**.308**	**.181**
		.1	**.579**	.213	.003	.0
	diffprivlib LR	.01	.507	.459	.299	.176
		.1	.547	.201	.003	.0
electricity	PrivaTree	.01	.563	**.417**	**.122**	**.026**
		.1	**.611**	.03	.0	.0
	diffprivlib LR	.01	.419	.31	.091	.019
		.1	.528	.026	.0	.0
eye_movements	PrivaTree	.01	**.507**	**.478**	**.376**	**.279**
		.1	.505	.277	.025	.001
	diffprivlib LR	.01	.487	.458	.361	.267
		.1	.506	.278	.025	.001
heloc	PrivaTree	.01	.517	.478	.347	.232
		.1	**.570**	.256	.01	.0
	diffprivlib LR	.01	.54	**.498**	**.362**	**.243**
		.1	.538	.242	.01	.0
house_16H	PrivaTree	.01	.567	**.513**	**.334**	**.194**
		.1	**.581**	.214	.003	.0
	diffprivlib LR	.01	.521	.472	.307	.179
		.1	.553	.204	.003	.0
jannis	PrivaTree	.01	.538	**.34**	**.054**	**.005**
		.1	**.653**	.007	.0	.0
	diffprivlib LR	.01	.51	.322	.051	.005
		.1	.552	.006	.0	.0
pol	PrivaTree	.01	.545	**.503**	**.365**	**.245**
		.1	**.549**	.247	.01	.0
	diffprivlib LR	.01	.535	.494	.359	.241
		.1	.526	.236	.01	.0

Table 7.2: 5-fold cross-validated mean test accuracy and poisoning accuracy guarantee against a percentage of poisoned samples on numerical datasets. Stronger privacy provides stronger poisoning robustness but comes at the cost of clean dataset accuracy. PrivaTree outperforms private logistic regression on this benchmark.

7.4 Results

We compare the performance of PrivaTree with regular decision trees from Scikit-learn [27] and 3 previous methods for training private decision trees: DiffPrivLib [13], BDPT [20] and DPGDF [28]. DiffPrivLib is a widely used Python library for differential privacy and implements several private machine learning models. Their decision tree implementation creates random decision nodes and uses all privacy budget to label leaves using the permute-and-flip mechanism. Since DiffPrivLib and Scikit-learn do not natively support categorical features, we encode them into integers. BDPT and DPGDF did not share their implementations, so we implemented these using primitives from DiffPrivLib. Since DPGDF only supports categorical variables, we run experiments as in the work of Borhan [16] and remove numerical features. BDPT only heuristically protects numerical feature values, so we compare it against PrivaTree*, a variant of PrivaTree in which we compute 10 bins by computing quantiles non-privately. These quantiles still heuristically protect privacy about the dataset's individual feature values, and the experiment allows us to compare the performance benefits of using quantiles instead of equal-width bins (PrivaTree).

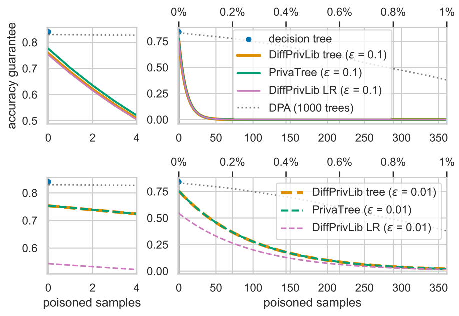
We also compare the performance of PrivaTree on poisoning robustness with differentially-private logistic regression and Deep Partition Aggregation (DPA). Previous works on poisoning robustness with differential privacy have used private logistic regression [19] as a robust interpretable model. We use the implementation by DiffPrivLib [13], which perturbs the optimal model parameters. Before training, we center the data and normalize it to unit variance, as this improves optimization. DPA trades interpretability and privacy for strong poisoning robustness guarantees by ensembling many models trained on distinct data subsets. We implement DPA with 1000 decision trees as tree-based models generally work better than neural networks for tabular data [29]. All experiments ran on a computer with 16GB of RAM and a 2 GHz Intel i5 processor with 4 cores.

7.4.1 Predictive performance

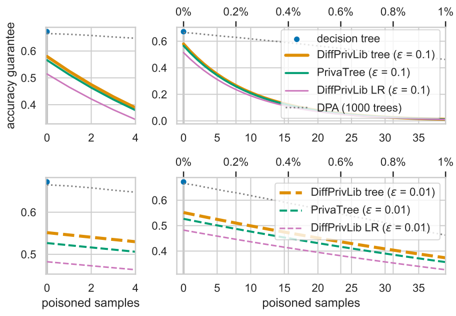
To compare PrivaTree to existing works, we evaluated performance using two well-known benchmarks. First, we experimented on 6 datasets from the UCI repository [30] that previous works tested on. However, these datasets are often small (*diabetes*), too easy to predict (*nursery*), or imbalanced (*adult*), which skews performance numbers. To complement this, we therefore also run experiments on the tabular data benchmark [29]. These datasets were chosen to be real-world, balanced, not too small, and not too simple. We removed rows with missing values and computed the public feature ranges based on the datasets. The categorical values are supplied by OpenML [31].

In Table 7.1, we present the accuracy scores on both benchmarks for trees of depth 4 computed with 5-fold stratified cross-validation at a privacy budget of $\epsilon = 0.1$. Results for other budgets are given in the appendix[2]. Since all private algorithms are based on the standard greedy algorithm for decision trees, the goal is to score similarly to those non-private trees. On almost all datasets PrivaTree outperforms BDPT, DPGDF, and DiffPrivLib or performs similarly. On *breast-w* and *vote*, however, DiffPrivLib sometimes performs better. This is because, on such small datasets, it is better to avoid spending the privacy budget on good decision nodes and instead spend all the budget on labeling leaves correctly. On av-

[2]`https://github.com/tudelft-cda-lab/PrivaTree`



(a) *adult* dataset (45222 samples, 14 features)



(b) *compas-two-years* (4966 samples, 11 features)

Figure 7.3: Guaranteed accuracy when varying the number of poisoned samples to up to 1% of the dataset size for trees of depth 4. A DPA ensemble of 1000 trees offers strong guarantees but is not interpretable, nor does it maintain privacy. For differentially-private learners, there is a trade-off between clean accuracy and the quality of the robustness guarantee.

erage, there is a 4% difference in accuracy score between PrivaTree using equal-width bins and PrivaTree* using non-private quantiles. BDPT has previously only been tested on numerical features with few unique values and fails to train accurate trees on the numerical tabular benchmark. In Figure 7.1, we visualize the average accuracy over 50 runs when varying the total privacy budget ϵ for depth 4 trees on the *adult* and *compas-two-years* datasets. Again, when ϵ is small compared to the dataset size, DiffPrivLib outperforms the other methods. However, when there is enough privacy budget to see value in choosing better decision nodes, PrivaTree improves over the other methods. On average, PrivaTree beats DiffPrivLib as when it wins, the score is significantly higher, while the score is only slightly lower when PrivaTree loses.

7.4.2 Poisoning robustness guarantees for tabular data

Recall from Section 7.3 that differentially private learners offer guarantees on the loss of accuracy incurred by attackers who poison the training dataset. In Figure 7.3, we visualized the guaranteed accuracy when varying the number of poisoned sampled in the dataset from 0% to 1% of the original train set size. These guarantees were determined by estimating the expected test accuracy with 50 random train-test splits and then computing the guarantees with Corollary 2. For comparison, we visualize the guarantee for DPA with an ensemble of 1000 depth 4 trees, but this method does not protect privacy and is not interpretable. The strength of the poisoning robustness guarantee for private learners is determined largely by the choice of ϵ and, to a lesser extent, the accuracy on the clean dataset. Therefore, there is an important trade-off between clean data accuracy and robustness guarantee.

Since previous works have considered the poisoning robustness guarantees of private logistic regression as an interpretable model, we compare against this method in more detail. In Table 7.2, we display the accuracy and guarantees at various poison levels for private logistic regression and PrivaTree. We evaluated the models on numerical datasets

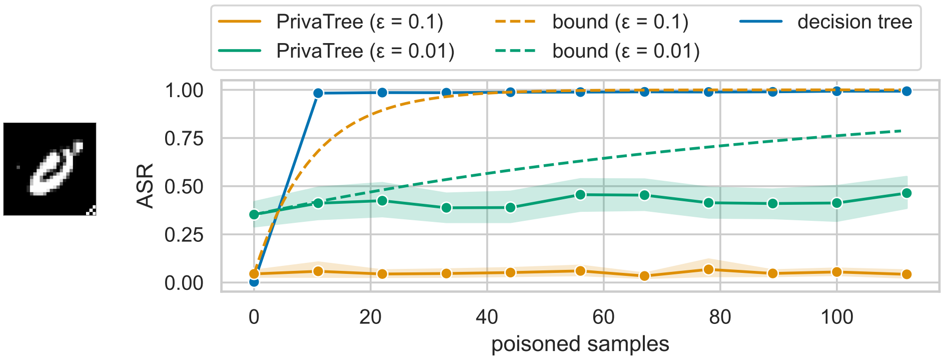


Figure 7.4: An adversary injects zeros with a trigger pattern to create a backdoor for class 1. (left) Poisoned sample of a 0 labeled as 1 with a trigger pattern in the bottom-right. (right) The attack success rate when varying the number of poisoned samples in MNIST 0 vs 1 out of 11,200 train samples. ϵ=0.01 offers a tighter bound than ϵ=0.1, but in practice, both values defend well against the backdoor attack on this dataset.

as this will prevent differences in performance due to the way categorical features are encoded for logistic regression and PrivaTree. Results on data with categorical features can be found in the appendix[3]. It is clear that there is a trade-off between accuracy and poison guarantee for both methods as guarantees at levels of 0.1% data poisoning are already always better when $\epsilon = 0.01$, whereas test accuracy is always better when $\epsilon = 0.1$. PrivaTrees outperform private logistic regression at equal privacy levels as these complex datasets likely contain non-linear patterns that are better captured by decision trees. Therefore, in the future, one should consider private decision trees as an alternative to private logistic regression when learning interpretable models with poisoning robustness guarantees.

7.4.3 Backdoor robustness on MNIST

To demonstrate the effectiveness of differentially private decision tree learners at mitigating data poisoning attacks, we evaluated backdoor attacks on the MNIST 0 vs 1 dataset. Specifically, we repeat the experiment from Badnets [32] in which the adversary adds a fixed trigger pattern to the bottom right corner of the image in an attempt to force zeros to be classified as ones. To achieve this, the adversary copies x zeros, adds the trigger pattern to these images, and adds the copies to the training set with label 1. An example of a zero with a trigger pattern is shown in Figure 7.4. To measure the robustness of models against the backdoor, we compute the Attack Success Rate (ASR), which is the percentage of test samples with label 0 that are predicted as 1 when the trigger pattern is added. In Figure 7.4, we plot the ASR of a regular decision tree and PrivaTrees with privacy budgets 0.1 and 0.01 and their bounds computed with Corollary 3 against a varying number of poisoned samples ranging between 0% to 1% of the dataset. All trees were trained on 50 train-test splits and had a depth of 4. With only 0.01% of the train set poisoned, regular decision

[3]`https://github.com/tudelft-cda-lab/PrivaTree`

trees already suffer from an ASR of almost 100%, whereas PrivaTrees, on average, stay at an ASR of under 20% for the entire range. While the bound for $\epsilon = 0.01$ is much tighter than the bound for $\epsilon = 0.1$, PrivaTree performs well in practice for both settings.

Strong privacy and robustness guarantees come at the cost of utility, i.e. clean dataset accuracy. The test accuracy scores without poisoned samples for the models in Figure 7.4 were 99.6% for the regular decision tree, 95.1% for PrivaTree with ϵ=0.1, and 77.7% for PrivaTree with ϵ=0.01.

7.5 Discussion

Some limitations can arise when applying PrivaTree to real-world scenarios. In our experiments, we compared the performance of various differentially-private decision tree learners on UCI data and the tabular data benchmark. While some UCI datasets are too easy, the tabular benchmark [29] was specifically curated so that decision trees alone do not easily score perfectly. For real use cases, data could be easier to classify. Also, while our code supports multiclass classification these benchmarks contain only binary classification tasks and so we have not evaluated models in this setting. It is worth noting that in the multiclass case methods such as logistic regression need to train a separate model for each class which hinders interpretability while decision trees still learn one tree.

As is typical, we assume that the range of numerical features, the set of possible categorical values, and the set of class labels are public knowledge. Additionally, we use the number of samples in the training set to select an efficient value for ϵ_{leaf} which assumes that we can publish information on the dataset size while some other works protect this value. One mitigation is to use rough estimates of the dataset size, e.g. only the order of magnitude.

Privacy and robustness in machine learning are important topics as models trained on user data are continuously deployed in the world. Differential privacy is a promising technique for this and we improve the performance of decision trees at high differential privacy levels. However, we want to warn against over-optimism as differential privacy is not a silver bullet for AI security. Engineers must take into account in which context models are deployed to decide what constitutes an acceptable privacy risk and must take into account what attributes are not protected by our method. Regarding poisoning robustness, it is also vital to understand the threat model that is being defended against to verify that the robustness guarantees for differentially-private learners apply. We hope that improvements in the privacy-utility trade-off for interpretable learners, such as the ones we propose, will increase the adoption of interpretable and private methods to improve the trustworthiness of machine learning systems.

7.6 Conclusion

In this paper, we proposed a new algorithm for training differentially private decision trees called PrivaTree. PrivaTree uses histograms with permute-and-flip for node selection, the permute-and-flip mechanism for leaf labeling, and a more efficient privacy budget distribution method to improve the privacy-accuracy trade-off. Our experiments on two benchmarks demonstrate that PrivaTree on average scores better than existing works at a fixed privacy budget. Moreover, we investigated the poisoning robustness guarantees for differentially-private learners and also applied this to the setting of backdoor attacks. On the MNIST 0 vs 1 task, differentially-private decision trees provide a fivefold reduction in attack success rate compared to decision trees trained without privacy. While our work makes progress in privacy budget allocation to improve the privacy-utility trade-off, follow-up work may further improve this trade-off in the very high privacy regime.

Bibliography

[1] Z. C. Lipton, *The mythos of model interpretability,* ACM Queue **16**, 30 (2018).

[2] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,* Nature machine intelligence **1**, 206 (2019).

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees* (Wadsworth, 1984).

[4] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[5] C. Dwork, *Differential privacy,* in *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, Lecture Notes in Computer Science, Vol. 4052, edited by M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener (Springer, 2006) pp. 1–12.

[6] M. Abadi, A. Chu, I. J. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, *Deep learning with differential privacy,* in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, edited by E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi (ACM, 2016) pp. 308–318.

[7] L. Breiman, *Random forests,* Mach. Learn. **45**, 5 (2001).

[8] J. H. Friedman, *Stochastic gradient boosting,* Computational statistics & data analysis **38**, 367 (2002).

[9] T. Chen and C. Guestrin, *XGBoost: A scalable tree boosting system,* in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[10] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, *Lightgbm: A highly efficient gradient boosting decision tree,* in *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017).

[11] M. Bojarski, A. Choromanska, K. Choromanski, and Y. LeCun, *Differentially- and non-differentially-private random decision trees,* CoRR **abs/1410.6973** (2014), 1410.6973 .

[12] S. Fletcher and M. Z. Islam, *Differentially private random decision forests using smooth sensitivity,* Expert Syst. Appl. **78**, 16 (2017).

[13] N. Holohan, S. Braghin, P. M. Aonghusa, and K. Levacher, *Diffprivlib: The IBM differential privacy library,* CoRR **abs/1907.02444** (2019), 1907.02444 .

[14] G. Jagannathan, K. Pillaipakkamnatt, and R. N. Wright, *A practical differentially private random decision tree classifier,* Trans. Data Priv. **5**, 273 (2012).

[15] A. Blum, C. Dwork, F. McSherry, and K. Nissim, *Practical privacy: the sulq framework,* in *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*, edited by C. Li (ACM, 2005) pp. 128–138.

[16] N. Borhan, *Budget allocation on differentially private decision trees and random forests*, Ph.D. thesis, University of Technology Sydney, Australia (2018).

[17] S. Fletcher and M. Z. Islam, *A differentially private decision forest,* in *Thirteenth Australasian Data Mining Conference, AusDM 2015, Sydney, Australia, August 2015*, CRPIT, Vol. 168, edited by K. Ong, Y. Zhao, M. G. Stone, and M. Z. Islam (Australian Computer Society, 2015) pp. 99–108.

[18] A. Friedman and A. Schuster, *Data mining with differential privacy,* in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, edited by B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang (ACM, 2010) pp. 493–502.

[19] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, *Differentially private empirical risk minimization.* Journal of Machine Learning Research **12** (2011).

[20] Z. Guan, X. Sun, L. Shi, L. Wu, and X. Du, *A differentially private greedy decision forest classification algorithm with high utility,* Comput. Secur. **96**, 101930 (2020).

[21] J. Gillenwater, M. Joseph, and A. Kulesza, *Differentially private quantiles,* in *International Conference on Machine Learning* (PMLR, 2021) pp. 3713–3722.

[22] A. Smith, *Privacy-preserving statistical estimation with optimal convergence rates,* in *Proceedings of the forty-third annual ACM symposium on Theory of computing* (2011) pp. 813–822.

[23] R. McKenna and D. R. Sheldon, *Permute-and-Flip: A new mechanism for differentially private selection,* Advances in Neural Information Processing Systems **33**, 193 (2020).

[24] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, *Anti-backdoor learning: Training clean models on poisoned data,* Advances in Neural Information Processing Systems **34**, 14900 (2021).

[25] D. Wu and Y. Wang, *Adversarial neuron pruning purifies backdoored deep models,* Advances in Neural Information Processing Systems **34**, 16913 (2021).

[26] Y. Ma, X. Zhu, and J. Hsu, *Data poisoning against differentially-private learners: Attacks and defenses,* arXiv preprint arXiv:1903.09860 (2019).

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, *Scikit-learn: Machine learning in python,* the Journal of machine Learning research **12**, 2825 (2011).

[28] B. Xin, W. Yang, S. Wang, and L. Huang, *Differentially private greedy decision forest,* in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019* (IEEE, 2019) pp. 2672–2676.

[29] L. Grinsztajn, E. Oyallon, and G. Varoquaux, *Why do tree-based models still outperform deep learning on tabular data?* arXiv preprint arXiv:2207.08815 (2022).

[30] D. Dua and C. Graff, *UCI machine learning repository,* (2017).

[31] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, *OpenML: networked science in machine learning,* SIGKDD Explorations **15**, 49 (2013).

[32] T. Gu, B. Dolan-Gavitt, and S. Garg, *Badnets: Identifying vulnerabilities in the machine learning model supply chain,* arXiv preprint arXiv:1708.06733 (2017).

II

Optimizing Decision Trees for **Sequential Decision Making**

8 Introduction to Decision Trees for Sequential Decision Making

When the predictions of machine learning models are used to make multiple sequential decisions in an environment, we can no longer rely on supervised learning algorithms. Particularly, the effect of previous predictions on future predictions is difficult to deal with. A popular model for such sequential decision-making problems is the Markov Decision Process (MDP). In an MDP, an agent can sequentially take actions depending on information about its current state, and the agent attempts to maximize the amount of reward received from the environment. For example, one can model solving a maze as an MDP, where the agent gets rewarded for escaping the maze. In this part, we consider learning algorithms for decision tree policies, i.e., decision trees that can be used to determine a series of sequential actions in uncertain environments. Decision tree policies allow users to interpret the behavior of an agent exactly.

8.1 Markov Decision Processes

Markov Decision Processes (MDPs) are useful for modeling non-deterministic optimization problems in which multiple sequential decisions must be made. They model an agent that is initialized in a certain state and can perform actions to traverse the state space and earn rewards. The goal is for the agent to find what action to take in each state to maximize the long-term sum of rewards. The model is formally specified by a tuple $\langle S, A, P, R, p_0 \rangle$ with states S, actions A, transition probabilities $P : S \times S \times A \to [0,1]$, reward values $R : S \times S \times A \to \mathbb{R}$ and initial probabilities $p_0 : S \to [0,1]$. When we solve an MDP, we usually mean that we find a function $\pi : S \to A$, also called a policy, that maps states to actions. In some MDPs, states are represented by a tuple of observations (features) that hold information about the state.

8.1.1 Solving MDPs

When solving MDPs, we generally discount future rewards in each step by a user-defined value of $0 < \gamma < 1$ to ensure that the optimal policy will generate a finite return. The most common approach for optimally solving MDPs is by using one of many dynamic programming variants [1]. In this dissertation, we use value iteration when we need to solve an MDP without limitations on the policy.

Value iteration finds a value V_s for each state s that holds the optimal expected discounted return for taking optimal greedy actions starting from that state. These values can be found by iteratively updating V_s until the Bellman equation [2]

$$V_s = \sum_{s'} P(s,s',a)R(s,s',a) + \sum_{s'} \gamma P(s,s',a)V_{s'}$$

is approximately satisfied. The optimal policy is not directly modeled but after solving one can find the optimal action a^* in state s by computing $a^* = \arg\max_a \sum_{s'} \gamma P(s,s',a)V_{s'}$. We will refer to this optimal solution found with value iteration as the *unrestricted* optimal solution, as the computed policy can be arbitrarily complex. When solving MDPs, there is always a deterministic policy that achieves the optimal expected return. However, when solving Partially Observable MDPs (POMDPs), in which the agent cannot determine exactly the state that they are in, it is possible that the only optimal policy is non-deterministic. Such a non-deterministic policy maps state observations to a probability distribution over the actions.

8.2 Reinforcement Learning

Reinforcement learning is the problem of finding a policy that maximizes the expected sum of rewards in an unknown Markov Decision Process (MDP) by repeatedly acting and observing the outcomes. Algorithms that solve this problem without explicitly modeling the MDP are called model-free, and there are two main styles of learning algorithms: Q-learning and policy gradient. In Q-learning, the agent attempts to learn the Q-value function, which maps states and actions to a value representing how good it is to take the action in the given state. In this work, we will compare to Deep Q-Networks [3], models that solve reinforcement learning problems by approximating the Q-value function with a neural network.

Policy gradient style algorithms [4] directly optimize a policy that maps states to action probabilities by running the policy in the environment, estimating the advantage of performing its predicted actions, and updating the policy to do more or less of those actions depending on the advantage scores. A major difference between Q-learning and policy gradient methods is that Q-learning usually deviates from its current policy to learn Q-values for the whole state space, while policy gradient methods only improve the policy in states that are reached by that policy.

8.2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [5] is an established policy gradient style method for reinforcement learning. The earliest policy gradient methods using function approximation [4] suffered from forgetting, where an update to the policy could change the policy excessively and thereby dramatically reduce performance. Methods such as Trust Region Policy Optimization (TRPO) [6] fixed this by constraining the Kullback–Leibler divergence of the policy before and after an update, however, due to that constraint TRPO is complex to implement. PPO prevents large policy updates by 'clipping' the objective values, i.e. projecting values onto a fixed number range. This makes PPO significantly easier to implement than TRPO. PPO maximizes the loss function L^{CLIP} for the policy where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the ratio of probabilities $\pi_\theta(a_t|s_t)$ assigned to the taken action a_t in state s_t between the updated and old policy:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]. \tag{8.1}$$

Here θ represents the model parameters, $\hat{A}_t$ the estimated advantage of environment interaction at time t and ϵ the clipping hyperparameter, which is typically set as $\epsilon = 0.2$. By taking the minimum of the clipped and unclipped terms, the agent is not rewarded for changes to the action probabilities beyond a ratio of $[1-\epsilon, 1+\epsilon]$. To compute the advantage estimates $\hat{A}_t$, PPO uses Generalized Advantage Estimation [7] and uses a separate neural network that estimates the value function by minimizing the squared error loss between observed and predicted returns. The value loss is also often clipped to prevent large updates. Often, one also adds a small term $cH(\pi(s_t))$ to the objective with $c = 0.01$; this stimulates exploration by promoting larger policy entropy.

8.2.2 Optimizing Decision Tree Policies

While there exists plenty of work optimizing decision trees for supervised targets, decision tree optimization for reinforcement learning has been less explored even though decision trees would offer better interpretability and verifiability properties than other policies. Many popular reinforcement learning techniques that use function approximation in Q-learning or policy gradient style algorithms require models to be differentiable. Such techniques are difficult to apply to inherently non-differentiable decision trees. In this work, we consider the problem of learning a decision tree as a policy in reinforcement learning settings. Other methods for interpretable or explainable methods for reinforcement learning are discussed in [8, 9].

Some methods have been proposed before that relax or restrict the decision tree learning problem to allow gradient updates. Policy tree [10] is a method that uses linear models as tree leaves together with a greedy splitting procedure to train decision tree policies with policy gradient techniques. While this algorithm trains well, the resulting tree model consists of multiple different linear models in the leaves, which makes it difficult to interpret the predictive behavior and can never backtrack once a decision node has been created. Similarly, Conservative Q-Improvement [11] greedily expands a decision tree to improve the Q-values, but the algorithm cannot update the tree structure multiple times. Differentiable decision trees [12] relax the property that each branch node selects one feature, that each leaf predicts a single action, and that only one branch is taken at a time, the resulting 'soft' decision tree is then optimizable with PPO. Although the soft decision tree can be optimized, the model usually incurs a significant drop in performance when discretizing into a 'crisp' decision tree. Paleja et al. [2022] improve this discretization error by using a different relaxation technique, but to achieve small performant trees, the algorithm uses linear functions in the leaves, which makes the models harder to interpret. Likmeta et al. [2020] propose fixing the branching nodes of the decision tree by using expert knowledge and optimizing the differentiable leaf values using gradient descent.

Iterative Bounding MDPs [15] is a method that extends known *factored* MDPs with tree-building actions such that neural networks can be used to optimize them. Since this method requires a specification of the complete MDP it cannot directly be applied to reinforcement learning. dtControl [16] converts controllers into decision trees for verification, but the resulting trees are usually too large to be interpreted. To the best of our knowledge, no published policy gradient-style algorithm can directly optimize 'hard' decision trees for reinforcement learning.

8.2.3 Imitation Learning (VIPER)

Instead of directly optimizing a decision tree, one can also try to extract a decision tree policy from a more complex teacher policy using imitation learning. These imitation learning algorithms turn reinforcement learning into a supervised learning problem for which we have successful decision tree learning algorithms [17, 18]. DAGGER [19] (dataset aggregation) is an algorithm that iteratively collects traces from the environment using its current policy and trains a supervised model on the union of the current and previous traces. Since DAGGER only uses information on the predicted action of the teacher policy, it ignores extra information on Q-values that modern Q-learning algorithms provide. VIPER [20] focuses on learning decision trees and improves on DAGGER by including Q-value information into the supervised learning objective. While VIPER generates significantly smaller decision trees than DAGGER, we will show that these trees are not yet optimal with respect to the trade-off in size and performance.

Bibliography

[1] M. L. Puterman, *Markov decision processes,* Wiley Series in Probability and Statistics (1994).

[2] R. Bellman, *A Markovian decision process,* Journal of mathematics and mechanics , 679 (1957).

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, *Human-level control through deep reinforcement learning,* nature **518**, 529 (2015).

[4] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, *Policy gradient methods for reinforcement learning with function approximation,* Advances in neural information processing systems **12** (1999).

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms,* arXiv preprint arXiv:1707.06347 (2017).

[6] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization,* in *International conference on machine learning* (PMLR, 2015) pp. 1889–1897.

[7] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, *High-dimensional continuous control using generalized advantage estimation,* arXiv preprint arXiv:1506.02438 (2015).

[8] C. Glanois, P. Weng, M. Zimmer, D. Li, T. Yang, J. Hao, and W. Liu, *A survey on interpretable reinforcement learning,* arXiv preprint arXiv:2112.13112 (2021).

[9] S. Milani, N. Topin, M. Veloso, and F. Fang, *A survey of explainable reinforcement learning,* arXiv preprint arXiv:2202.08434 (2022).

[10] U. D. Gupta, E. Talvitie, and M. Bowling, *Policy tree: Adaptive representation for policy gradient,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29 (2015).

[11] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, *Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy,* arXiv preprint arXiv:1907.01180 (2019).

[12] A. Silva, M. Gombolay, T. Killian, I. Jimenez, and S.-H. Son, *Optimization methods for interpretable differentiable decision trees applied to reinforcement learning,* in *International conference on artificial intelligence and statistics* (PMLR, 2020) pp. 1855–1865.

[13] R. R. Paleja, Y. Niu, A. Silva, C. Ritchie, S. Choi, and M. C. Gombolay, *Learning interpretable, high-performing policies for autonomous driving,* Robotics: Science and Systems XVIII (2022).

[14] A. Likmeta, A. M. Metelli, A. Tirinzoni, R. Giol, M. Restelli, and D. Romano, *Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving,* Robotics and Autonomous Systems **131**, 103568 (2020).

[15] N. Topin, S. Milani, F. Fang, and M. Veloso, *Iterative Bounding MDPs: Learning interpretable policies via non-interpretable methods,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35 (2021) pp. 9923–9931.

[16] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínskỳ, M. Weininger, and M. Zamani, *dtcontrol: Decision tree learning algorithms for controller representation,* in *Proceedings of the 23rd international conference on hybrid systems: Computation and control* (2020) pp. 1–7.

[17] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees* (Taylor & Francis, 1984).

[18] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[19] S. Ross, G. Gordon, and D. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning,* in *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (JMLR Workshop and Conference Proceedings, 2011) pp. 627–635.

[20] O. Bastani, Y. Pu, and A. Solar-Lezama, *Verifiable reinforcement learning via policy extraction,* Advances in neural information processing systems **31** (2018).

9 Optimal Decision Tree Policies for Markov Decision Processes

Interpretability of reinforcement learning policies is essential for many real-world tasks, but learning such interpretable policies is a hard problem. Particularly, rule-based policies such as decision trees and rules lists are difficult to optimize due to their non-differentiability. While existing techniques can learn verifiable decision tree policies, there is no guarantee that the learners will generate a policy that performs optimally. In this work, we study the optimization of size-limited decision trees for Markov Decision Processes (MPDs) and propose OMDTs: Optimal MDP Decision Trees. Given a user-defined size limit and MDP formulation, OMDT directly maximizes the expected discounted return for the decision tree using Mixed-Integer Linear Programming. By training optimal tree policies for different MDPs, we empirically study the optimality gap for existing imitation learning techniques and find that they perform sub-optimally. We show that this is due to an inherent shortcoming of imitation learning, namely that complex policies cannot be represented using size-limited trees. In such cases, it is better to directly optimize the tree for expected return. While there is generally a trade-off between the performance and interpretability of machine learning models, we find that on small MDPs, depth 3 OMDTs often perform close to optimally.

This chapter is based on *D. Vos and S. Verwer. Optimal Decision Tree Policies for Markov Decision Processes, IJCAI 2023* [1].

9.1 Introduction

Advances in reinforcement learning using function approximation have allowed us to train powerful agents for complex problems such as the games of Go and Atari [2]. Policies learned using function approximation often use neural networks, making them impossible for humans to understand. Therefore, reinforcement learning is severely limited for applications that involve high-stakes decisions and require the user to trust the learned policy.

Recent work has focused on *explaining* opaque models such as neural networks by attributing prediction importance to the input features [3, 4]. However, these explanation methods cannot capture the full complexity of their models, which can mislead users when attempting to understand the predictions [5]. Concurrently, there has been much work on *interpretable* machine learning in which the model learned is limited in complexity to the extent that humans can understand the complete model. Particularly decision trees have received much attention as they are simple models that are capable of modeling non-linear behavior [6].

Decision trees are difficult to optimize as they are non-differentiable and discontinuous. Previous works have used different strategies to overcome the hardness of optimizing trees: using assumptions or relaxations to make the trees differentiable [7–9], reformulating the MDP into a meta-MDP that exclusively models decision tree policies [10] or extracting trees from a complex teacher [11]. While these methods are increasingly successful in training performant trees they do not offer guarantees on this performance.

Our work takes a step towards bridging the gap between the fields of optimal decision trees and reinforcement learning. Existing formulations for optimal decision trees assume a fixed training set with independent samples. This cannot be used in a dynamic setting where actions taken in one state influence the best actions in others. Instead, we formulate the problem of solving a Markov Decision Process (MDP) using a policy represented by a size-limited decision tree (see Figure 9.1) in a single MILP. We link the predictions of the decision tree policy to the state-action frequencies in the dual linear program for solving MDPs. The dual allows us to reason over policies explicitly, which results in a more efficient formulation. Our formulation for Optimal MDP Decision Trees, OMDTs, optimizes a decision tree policy for a given MDP and a tree size limit. OMDT produces increasingly performant policies as runtime progresses and eventually proves its policy's optimality under the size constraint.

Existing methods for training size-limited decision trees in reinforcement learning such as VIPER [11] make use of imitation learning, where a student tries to learn from a powerful teacher policy. We compare the performance of OMDT and VIPER on a variety of MDPs. Interestingly, we show that, when training interpretable size-limited trees, imitation learning performs significantly worse as the capacity of the learned decision tree is wasted on parts of the state space that are never reached by the policy. Moreover, VIPER cannot prove optimality even if it identifies the optimal solution. Regarding the performance-interpretability trade-off, we show that decision trees of 7 decision nodes are enough to perform close to unrestricted optimal policies in 8 out of 13 environments. Such trees are orders of magnitude smaller than size-unrestricted trees created by methods that replicate the unrestricted policy, such as dtcontrol [12].

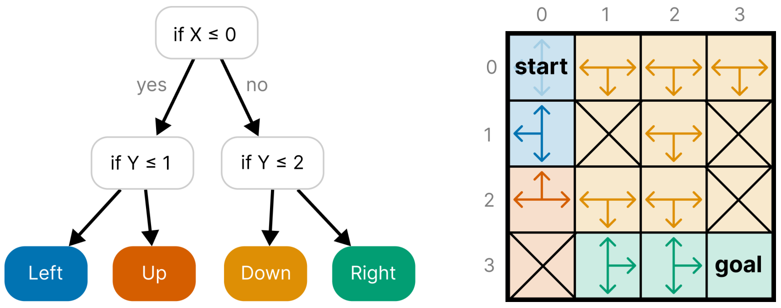


Figure 9.1: Depth 2 OMDT on the stochastic Frozenlake 4x4 environment. OMDT proves that no better depth 2 decision tree policy exists (discounted return 0.37 with $\gamma = 0.99$).

9.2 OMDT: Optimal MDP Decision Trees

We introduce OMDTs, optimal MDP decision trees, as a first step in bridging the gap between optimal decision trees for supervised and reinforcement learning. OMDT is a Mixed-Integer Linear Programming formulation that encodes the problem of identifying a decision tree policy that achieves maximum discounted return given a user-defined MDP and tree size limit. Our formulation can be solved using one of many available solvers, in this work we use the state-of-the-art solver Gurobi[1].

Intuitively the OMDT formulation consists of two parts: a (dual) linear programming formulation for solving MDPs and a set of constraints that limits the set of feasible policies to decision trees. Figure 9.2 summarizes OMDT's formulation in natural language. All the notation used in OMDT is summarized in Table 9.1.

9.2.1 Constraints

It is well known that MDPs can be solved using linear programming, the standard linear program is [13]:

$$\begin{aligned} \text{min.} \quad & \sum_s p_0(s) V_s \\ \text{s.t.} \quad & V_s - \sum_{s'} \gamma P(s,s',a) V_{s'} \geq \sum_{s'} P(s,s',a) R(s,s',a), \;\; \forall s,a \end{aligned}$$

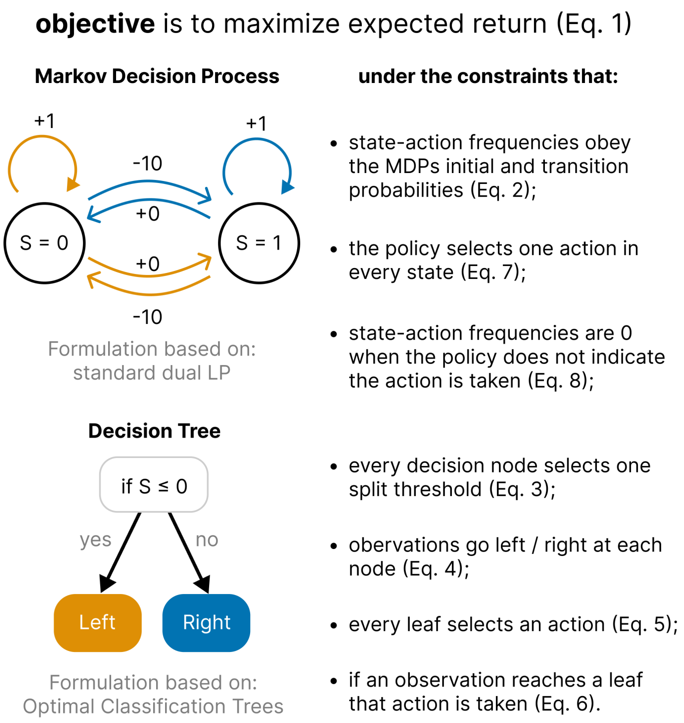
It is not easy to constrain the policy in this formulation because it reasons abstractly over policies, i.e. by reasoning over the policy's state values. To create a formulation for decision tree policies, we resort to the standard dual program:

$$\text{max.} \quad \sum_s \sum_a x_{s,a} \sum_{s'} P(s,s',a) R(s,s',a) \tag{9.1}$$

$$\text{s.t.} \quad \sum_a x_{s,a} - \sum_{s'} \sum_a \gamma P(s',s,a) x_{s',a} = p_0(s), \;\; \forall s \tag{9.2}$$

This program uses a measure $x_{s,a}$ of how often the agent takes action a in state s. This allows us to add efficient constraints that control the policy of the agent. Intuitively the pro-

[1]https://www.gurobi.com/


Markov Decision Process
+1
+1
-10
+0
+0
-10
S = 0
S = 1
Formulation based on: standard dual LP
Decision Tree
if S ≤ 0
yes
no
Left
Right
Formulation based on: Optimal Classification Trees

objective is to maximize expected return (Eq. 1)

under the constraints that:

- state-action frequencies obey the MDPs initial and transition probabilities (Eq. 2);
- the policy selects one action in every state (Eq. 7);
- state-action frequencies are 0 when the policy does not indicate the action is taken (Eq. 8);
- every decision node selects one split threshold (Eq. 3);
- obervations go left / right at each node (Eq. 4);
- every leaf selects an action (Eq. 5);
- if an observation reaches a leaf that action is taken (Eq. 6).

Figure 9.2: Overview of OMDT's formulation. We maximize the discounted return in an MDP under the constraint that the policy is represented by a size-limited decision tree.

gram maximizes the rewards $\sum_{s'} P(s,s',a)R(s,s',a)$ weighted by this $x_{s,a}$. The constraints enforce that the frequency by which a state is exited is equal to the frequency that the agent is initialized in the state $p_0(s)$ or returns to it following the discounted transition probabilities $\gamma P(s,s',a)$.

To enforce the policy to be a size-limited decision tree, we will later constrain the $x_{s,a}$ values to only be non-zero when the agent is supposed to take action a in state s according to a tree policy. We will first introduce the variables and constraints required to model the decision tree constraints.

Modeling Decision Nodes

Our decision tree formulation is roughly based on OCT [14] and ROCT [15], MILP formulations for optimal (OCT) and robust (ROCT) classification trees. In these formulations, the shape of the decision tree is fixed. Like ROCT, we describe a decision node m by binary threshold variables $b_{m,j,k}$, indicating whether the kth threshold of feature j is chosen.[2] Unlike ROCT, we only allow one of these variables to be true among all features and possible

[2]In practice, the possible values for threshold k depends on the chosen feature j. We do not model this for the convenience of notation.

thresholds:

$$\sum_j \sum_k b_{m,j,k} = 1, \quad \forall m \tag{9.3}$$

We follow paths through the tree to map observations to leaves. In each node m we decide the direction $d_{s,m}$ that the observation of state s takes (left=0 or right=1 of the threshold k). ROCT uses two variables per state-node pair to model directions $d_{s,m}$ to account for perturbations in the observations. Since we are optimizing for an MDP without uncertainty in the observations, we only require one variable $d_{s,m}$ per state-node pair.

We further improve over ROCT by determining $d_{s,m}$ using only one constraint per state-node pair instead of a separate constraint per state-node-feature triple. For this, we pre-compute a function side(s, j, k) which indicates for each feature-threshold pair (j, k) and every observation s the side of k that s is on for feature j (left=0 or right=1), i.e. whether $X_{sj} > k$ holds. This formulation is not limited to the predicates '$\leq$' or '$>$', however, and can be easily extended to other predicates in the pre-computation of side(s, j, k). The following then forces the direction $d_{s,m}$ to be equal to the direction of the indicated threshold:

$$d_{s,m} = \sum_j \sum_k \text{side}(s, j, k)\, b_{m,j,k}, \quad \forall s, m \tag{9.4}$$

The variables $d_{s,m}$ represent the direction of an observation's path at a decision node. Together, the d variables allow us to follow an observation's path through the tree, which we use to identify the leaf that it reaches. Important in this formulation, compared to existing binary encodings, is that it requires no big-M constraints to describe these paths. This makes the relaxation stronger, and therefore, the solver gives much better bounds than using the big-M style formulations from ROCT.

Modeling Policy Actions

Decision leaves only have one set of binary decision variables: $c_{t,a}$ encoding whether or not leaf t predicts action a. We want each leaf to select exactly one action:

$$\sum_a c_{t,a} = 1, \quad \forall t \tag{9.5}$$

As mentioned before, we can follow an observation's path through the tree by using their $d_{s,m}$ path variables. One can linearize an implication of a conjunction of binary variables as follows:

$$\begin{aligned} & x_1 \wedge x_2 \wedge \ldots \wedge x_n \implies y \\ \equiv\ & x_1 + x_2 + \ldots + x_n - n + 1 \leq y \end{aligned}$$

If an observation reaches leaf t and the leaf predicts action a, then we want to force the policy $\pi_{s,a}$ to take that action in the associated state s. Using the aforementioned equivalence, we add the constraint:

$$\sum_{m \in A_l(t)} (1 - d_{s,m}) + \sum_{m \in A_r(t)} d_{s,m} + c_{t,a} - |A(t)| \leq \pi_{s,a}, \quad \forall s, a, t \tag{9.6}$$

Name	Kind	Description
$b_{m,j,k}$	bin.	Tree uses feat. j and threshold k in node m
$c_{t,a}$	bin.	Tree selects action a in leaf t
$d_{s,m}$	bin.	Observation of s goes left / right in node m
$\pi_{s,a}$	bin.	Policy takes action a in state s
$x_{s,a}$	cont.	Frequency of action a taken in state s
$P(s,s',a)$	const.	Probability of transition $s \to s'$ with action a
$R(s,s',a)$	const.	Reward for transition $s \to s'$ with action a
$p_0(s)$	const.	Probability of starting in state s
γ	const.	Discount factor
X_{ij}	const.	Feature j's value of observation i
side(s,j,k)	const.	Side state s is on for thresh. k and feat. j
$a \in A$	set	Set of actions in MDP
$s \in S$	set	Set of states in MDP
$i{=}1..\lvert S\rvert$	set	Observation and state indices
$j \in J$	set	Set of feature indices
$k = 1..K$	set	Indices of all possible feature thresholds
$m \in \mathcal{T}_D$	set	Set of decision nodes in the tree
$t \in \mathcal{T}_L$	set	Set of leaves in the tree
$A(t)$	set	Set of ancestors of leaf t
$A_l(t)$	set	... that have t in their left path
$A_r(t)$	set	... that have t in their right path

Table 9.1: Summary of notation used in OMDT.

This constraint forces the agent to take the action indicated by the leaf. To prevent the agent from taking other actions that were not indicated, we force it to only take a single action in each state (giving a deterministic policy):

$$\sum_a \pi_{s,a} = 1, \quad \forall s \tag{9.7}$$

Now we have indicators $\pi_{s,a}$ that mark what action is taken by the agent. To link this back to the MDP linear programming formulation that we use to optimize the policy, we set the $x_{s,a}$ variables. We need to set $x_{s,a} = 0$ if $\pi_{s,a} = 0$, else $x_{s,a}$ should be unbounded. We encode this using a big-M formulation:

$$x_{s,a} \leq M\pi_{s,a}, \quad \forall s, a \tag{9.8}$$

M should be chosen as small as possible but larger or equal to the largest value that $x_{s,a}$ can take. We use the fact that we are optimizing the MDP using discount factor γ to compute an upper bound on $x_{s,a}$ and set $M = \frac{1}{1-\gamma}$, proof is given in the appendix[3].

[3]https://arxiv.org/abs/2301.13185

9.2.2 Complete Formulation

The runtime of MILP solvers grows worst-case exponentially with respect to formulation size, so it is important to limit the scale of the formulation. The number of variables in our formulation grows with $\mathcal{O}(|S||J||\mathcal{T}_D|+|A||\mathcal{T}_L|+|S||A|)$ which follows from their indices in Table 9.1. The number of constraints grows with the order $\mathcal{O}(|S||\mathcal{T}_D|+|S||A||\mathcal{T}_L|)$ as it is dominated by the constraints that determine $d_{s,m}$ at each node (Equation 9.4) and constraints that force $\pi_{s,a}$ according to the tree (Equation 9.6). We summarize OMDT below:

$$\max \quad \sum_s \sum_a x_{s,a} \sum_{s'} P(s,s',a)R(s,s',a) \tag{9.1}$$

s.t.

$$\sum_a x_{s,a} - \sum_{s'} \sum_a \gamma P(s',s,a)x_{s',a} = p_0(s), \forall s \tag{9.2}$$

$$\sum_j \sum_k b_{m,j,k} = 1, \forall m \tag{9.3}$$

$$d_{s,m} = \sum_j \sum_k \text{side}(s,j,k)\, b_{m,j,k}, \forall s,m \tag{9.4}$$

$$\sum_a c_{t,a} = 1, \forall t \tag{9.5}$$

$$\sum_{m\in A_l(t)} (1-d_{s,m}) + \sum_{m\in A_r(t)} d_{s,m} + c_{t,a} - |A(t)| \leq \pi_{s,a}, \forall s,a,t \tag{9.6}$$

$$\sum_a \pi_{s,a} = 1, \forall s \tag{9.7}$$

$$x_{s,a} \leq M\pi_{s,a}, \forall s,a \tag{9.8}$$

9.3 Results

We present experiments comparing the performance of OMDTs with VIPER and dtcontrol. Viper uses imitation learning to extract a size-limited decision tree from a teacher policy, and dtcontrol learns an unrestricted tree that exactly copies the teacher's behavior. To provide a fair comparison we have trained VIPER and dtcontrol with an unrestricted optimal teacher by first solving the MDP with value iteration and then extracting all Q values, both methods ran with default parameters. We also implemented and ran experiments on interpretable Differentiable Decision Trees [8] but excluded these models from our analysis as they did not outperform a random policy. The full code for OMDT and our experiments can be found on GitHub[4]. All of our experiments ran on a Linux machine with 16 Intel Xeon CPU cores and 72 GB of RAM total and used Gurobi 10.0.0 with default parameters. Each method ran on a single CPU core.

9.3.1 Environments

For comparison we implemented 13 environments based on well-known MDPs from the literature, the sizes of these MDPs are given in Table 9.2. All MDPs were pre-processed

[4]`https://github.com/tudelft-cda-lab/OMDT`

such that states that are unreachable from the initial states are removed. We briefly describe the environments below but refer to the appendix[5] for complete descriptions.

In *3d_navigation*, the agent controls a robot in a 5×5×5 world and attempts to reach from start to finish, with each voxel having a chance to make the robot disappear. *blackjack* is a simplified version of the famous casino game where we assume an infinite-sized deck and only the actions 'skip' or 'hit'. *frozenlake* is a grid world where the agent attempts to go from start to finish without falling into holes, actions are stochastic so the agent will not always move in the intended direction (e.g. the action 'up' will only not send the agent 'down'). *inventory management* models a company that has to decide how many items to order to maximize profit while minimizing cost. *system_administrator* refers to a computer network where computers randomly crash, and an administrator has to decide which computer to reboot. A crashed computer has an increased probability of crashing a neighboring computer. *tictactoe_vs_random* is the well-known game of tic-tac-toe when played against an opponent that makes random moves. In *tiger_vs_antelope*, the agent attempts to catch an antelope that randomly jumps away from the tiger in a grid world. *traffic_intersection* describes a perpendicular intersection where traffic flows in at different rates, and the operator decides when to switch the traffic lights. *xor* is an MDP constructed with states randomly distributed on a plain, the agent gets 1 reward for taking the action according to an XOR function and -1 for a mistake. The XOR problem is notoriously difficult for greedy decision tree learning algorithms.

9.3.2 Performance-Interpretability Trade-off

It is often assumed that there is a trade-off in the performance and interpretability of machine learning models [16], since interpretable models necessarily lack complexity, but this assumption is not always true [5]. We aim to answer whether the performance-interpretability trade-off occurs in a variety of MDPs by training size-limited decision trees and comparing their performance to the optimal solutions that were not restricted in complexity. We visualize the normalized return of depth 3 OMDTs and unrestricted dtcontrol trees in Figure 9.3. Returns were normalized such that 0 corresponds to a random policy and 1 to an optimal one. Since small deterministic decision tree policies are limited in the number of distinct actions, an optimal tree can perform worse than a random policy. Experiments were repeated 3 times, and runs were limited to 2 hours. We consider an OMDT optimal when the relative gap between its objective and bound is proven to be less than 0.01%.

While it is debatable what the precise size limits are for decision trees to be interpretable [6], we use trees of depth 3, which implies that a tree has at most 8 leaves. Note that this also limits the number of distinct actions in the policy to 8. We find that in all environments, OMDTs of depth 3 improve on the performance of random policies, and in 8 out of 13 environments, the policy gets close to optimal. Decision trees trained with dtcontrol always achieve the optimal normalized return of 1 since they exactly mimic the optimal policy. However, dtcontrol produces large trees that are not interpretable to humans. When run on 3d_navigation, for example, dtcontrol produces a tree of 68 decision nodes, which is very complex for humans to understand. OMDT produces a tree of 7 decision nodes that perform equally well.

[5]`https://arxiv.org/abs/2301.13185`

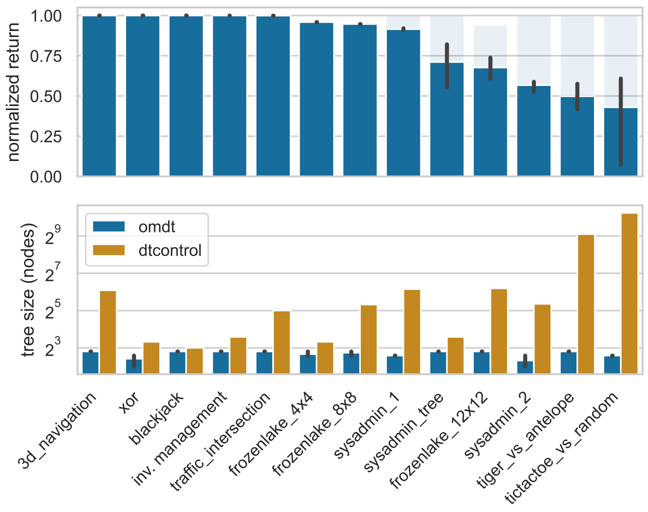


Figure 9.3: (top) Normalized return and bounds for OMDT trees of depth 3, optimal policies score 1 while uniform random policies score 0. (bottom) Log of tree sizes for OMDT (maximum depth 3) and dtcontrol. Dtcontrol always produces an optimal policy but the trees are orders of magnitude larger than OMDT.

Overall, our results demonstrate that for small environments there is no performance-interpretability trade-off: simple policies represented by size-limited trees perform approximately as well as the unrestricted optimal policy.

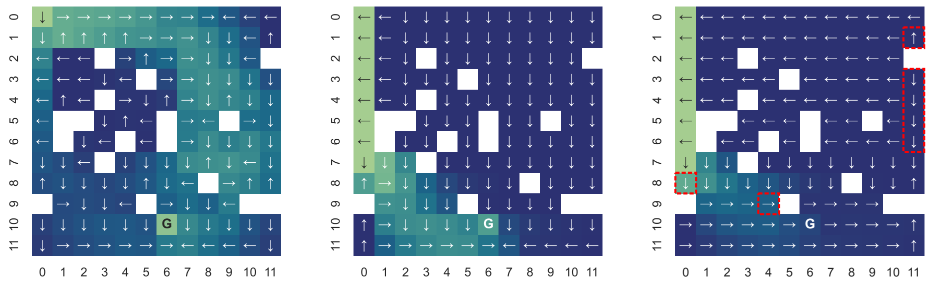
9.3.3 Direct Optimization versus Imitation Learning

The above conclusion holds when the policy is learned to optimality under the constraint that it has to be a small tree, e.g., using OMDT. Techniques such as VIPER enforce the size constraint but aim to imitate the unrestricted optimal policy. We now show that this comes at a cost when the unrestricted policy is too complex to be represented using a small tree.

VIPER trains trees by imitating high Q values of the optimal policies, while OMDT directly maximizes expected return. In Table 9.2, we list the normalized return (0 for random policies, 1 for optimal policies) for VIPER and OMDT with respectively 5 minutes and 2 hours of runtime. After 5 minutes, OMDT improves performance over random policies but often needs more time to improve over VIPER. After 2 hours OMDT's policies win on 11 out of 13 environments. For instances with large state space such as tictactoe_vs_random, OMDT needs more than 2 hours to improve over VIPER.

Shortcomings of Imitation Learning

Overall, given sufficient runtime, OMDT produces better policies than VIPER. This cannot be easily solved by giving VIPER more runtime but is an inherent problem of imitation learning. To illustrate this, we investigate the results on the frozenlake MDPs as Table 9.2 demonstrates that imitation learning can perform far from optimal in these environments.



(a) Optimal: 92% success (b) OMDT (depth 3): 66% success (c) VIPER (depth 3): 11% success

Figure 9.4: Paths taken on 10,000 Frozenlake_12x12 runs. The agent starts at (0, 0) and attempts to reach the goal tile 'G' while avoiding holes. Actions are indicated by arrows and are somewhat stochastic, i.e. an action of 'up' will send the agent 'left', 'up', or 'right' (but never down) with equal probability. VIPER fails to produce a good policy because it spends the capacity of its tree mimicking parts of the complex teacher policy that its simple student policy will never reach. OMDT achieves a greater success rate by directly optimizing a simple policy.

In theory, imitation learning performs optimally in the limit [17], but this result requires the student policy to be as expressive as the teacher policy. This is not the case for size-limited decision trees. When VIPER learns a policy for frozenlake_12x12 it tries to imitate a complex policy using a small tree that cannot represent all teacher policy actions. This results in VIPER spending capacity of its decision tree on parts of the state space that will never be reached under its student policy. In Figure 9.4, we visualize the paths that the agents took on 10,000 runs and indicate the policies with arrows. VIPER creates leaves that control action in the right section of the grid world (indicated in red). The optimal teacher policy often visits this section, but the simple student does not. By directly optimizing a decision tree policy using OMDT, the policy spends its capacity on parts of the state space that it actually reaches. As a result, VIPER cannot prevent actions that send its agent into holes on the left part of the grid world (indicated in red). OMDT actively avoids these.

9.3.4 Runtime

Runtime for solving Mixed-Integer Linear Programming formulations scales worst-case exponentially, which makes it important to understand how solvers operate on complex formulations such as OMDT. We solved OMDTs for a depth of 3 for a maximum of 2 hours and reported the results in Table 9.2. The table compares the runtimes of VIPER and solving OMDT to optimality. If the solver does not prove optimality within 2 hours, we denote it as 'timeout'. We also denote the number of possible decision tree policies computed as: $|\mathcal{T}_B|^{\text{possible splits}} \times |\mathcal{T}_L|^{|A|}$. It estimates the number of possible decision tree policies and shows that enumerating trees with brute force is intractable.

OMDT solves a simple environment such as Frozenlake 4x4 (16 states, 4 actions) to optimality within 2 seconds, but runtime grows for larger environments such as inventory management (101 states, 100 actions), which took an average of 2533 seconds. VIPER needs roughly 2250 seconds of runtime for every MDP and runs significantly faster on some MDPs. This is because VIPER spends much time evaluating policies on the environment, and some environments quickly reach terminal states, which results in short episodes. While OMDT was able to prove optimality on only 7 out of 13 environments within 2 hours, OMDT found good policies before this time on 12 out of 13 environments.

			normalized return			MILP			runtime (s)	
MDP	$\lvert S\rvert$	$\lvert A\rvert$	VIPER	OMDT 5 mins.	OMDT 2 hrs.	vars.	constrs.	trees	VIPER	OMDT optimal
3d_navigation	125	6	**1.00** ±.00	.81 ±.10	**1.00** ±.00	2,528	7,890	10^{14}	2,090 ±55	315 ±89
blackjack	533	2	**1.00** ±.00	**1.00** ±.00	**1.00** ±.00	6,187	14,406	10^{14}	2,248 ±27	408 ±85
frozenlake_4x4	16	4	.67 ±.00	**.96** ±.00	**.96** ±.00	328	735	10^{10}	74 ±3	2 ±0
frozenlake_8x8	64	4	.83 ±.06	**.95** ±.00	**.95** ±.00	1,104	2,895	10^{13}	178 ±5	98 ±30
frozenlake_12x12	144	4	.19 ±.09	.63 ±.03	**.68** ±.04	2,360	6,495	10^{14}	196 ±38	timeout
inv. management	101	100	**1.00** ±.00	.37 ±.37	**1.00** ±.00	22,414	91,824	10^{30}	2,254 ±86	2,533 ±540
sysadmin_1	256	9	.88 ±.01	.85 ±.01	**.92** ±.00	6,584	23,055	10^{14}	2,265 ±37	timeout
sysadmin_2	256	9	**.59** ±.00	.23 ±.06	.58 ±.01	6,584	23,055	10^{14}	2,257 ±7	timeout
sysadmin_tree	128	8	.57 ±.04	.48 ±.07	**.70** ±.09	3,106	10,383	10^{13}	2,136 ±72	timeout
tictactoe_vs_rand.	2,424	9	**.80** ±.01	-.06 ±.00	.43 ±.18	61,239	218,175	10^{20}	21 ±3	timeout
tiger_vs_antelope	626	5	-.10 ±.02	-.17 ±.19	**.52** ±.03	10,850	33,819	10^{15}	490 ±243	timeout
traffic_intersec.	361	2	.98 ±.00	.99 ±.00	**1.00** ±.00	4,127	9,762	10^{11}	2,188 ±121	1,219 ±177
xor	200	2	.34 ±.06	**1.00** ±.00	**1.00** ±.00	5,016	5,415	10^{21}	1,999 ±123	50 ±0

Table 9.2: Comparison of depth 3 trees trained with VIPER and OMDT on 13 MDPs, experiments were repeated 3 times, means and standard errors are given. All runs were limited to 2 hours. OMDT solves some MDPs in 5 minutes but significantly improves when given 2 hours of runtime. While 2 hours are enough for OMDT to achieve greater or equal scores to VIPER in most MDPs, OMDT needs more time to outperform VIPER on the large *tictactoe* MDP. OMDT was able to identify the optimal size-limited tree and prove its optimality on 7 MDPs.

9.4 Conclusion

We propose OMDT, a mixed-integer linear programming formulation for training optimal size-limited decision trees for Markov Decision Processes. Our results show that for simple environments such as blackjack, we do not have to trade off interpretability for performance: OMDTs of depth 3 achieve near-optimal performance. On Frozenlake 12x12, OMDT outperforms VIPER by more than 100%.

OMDT sets a foundation for extending supervised optimal decision tree learning techniques to the reinforcement learning setting. Still, OMDT requires a full specification of the Markov Decision Process. Imitation learning techniques such as VIPER can instead also learn from a simulation environment. Therefore, future work should focus on closing the gap between the theoretical bound supplied by OMDT and the practical performance achieved by algorithms that require only simulation access to optimize interpretable decision tree policies in reinforcement learning. Additionally, future work can incorporate factored MDPs into OMDT's formulation to scale up to larger state spaces.

Bibliography

[1] D. Vos and S. Verwer, *Optimal decision tree policies for markov decision processes,* in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, edited by E. Elkind (International Joint Conferences on Artificial Intelligence Organization, 2023) pp. 5457–5465, main Track.

[2] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, *Mastering Atari, Go, chess and shogi by planning with a learned model,* Nature **588**, 604 (2020).

[3] M. T. Ribeiro, S. Singh, and C. Guestrin, *"Why should I trust you?" Explaining the predictions of any classifier,* in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016) pp. 1135–1144.

[4] S. M. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions,* Advances in neural information processing systems **30** (2017).

[5] C. Rudin, *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,* Nature Machine Intelligence **1**, 206 (2019).

[6] Z. C. Lipton, *The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery.* Queue **16**, 31 (2018).

[7] U. D. Gupta, E. Talvitie, and M. Bowling, *Policy tree: Adaptive representation for policy gradient,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29 (2015).

[8] A. Silva, M. Gombolay, T. Killian, I. Jimenez, and S.-H. Son, *Optimization methods for interpretable differentiable decision trees applied to reinforcement learning,* in *International conference on artificial intelligence and statistics* (PMLR, 2020) pp. 1855–1865.

[9] A. Likmeta, A. M. Metelli, A. Tirinzoni, R. Giol, M. Restelli, and D. Romano, *Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving,* Robotics and Autonomous Systems **131**, 103568 (2020).

[10] N. Topin, S. Milani, F. Fang, and M. Veloso, *Iterative Bounding MDPs: Learning interpretable policies via non-interpretable methods,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35 (2021) pp. 9923–9931.

[11] O. Bastani, Y. Pu, and A. Solar-Lezama, *Verifiable reinforcement learning via policy extraction,* Advances in neural information processing systems **31** (2018).

[12] P. Ashok, M. Jackermeier, P. Jagtap, J. Křetínskỳ, M. Weininger, and M. Zamani, *dtcontrol: Decision tree learning algorithms for controller representation,* in *Proceedings of the 23rd international conference on hybrid systems: Computation and control* (2020) pp. 1–7.

[13] M. L. Puterman, *Markov decision processes,* Wiley Series in Probability and Statistics (1994).

[14] D. Bertsimas and J. Dunn, *Optimal classification trees,* Machine Learning **106**, 1039 (2017).

[15] D. Vos and S. Verwer, *Robust optimal classification trees against adversarial examples,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36 (2022) pp. 8520–8528.

[16] D. Gunning and D. Aha, *DARPA's explainable artificial intelligence (XAI) program,* AI magazine **40**, 44 (2019).

[17] S. Ross, G. Gordon, and D. Bagnell, *A reduction of imitation learning and structured prediction to no-regret online learning,* in *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (JMLR Workshop and Conference Proceedings, 2011) pp. 627–635.

10

DT-PPO: Interpretable Proximal Policy Optimization using Decision Trees

Reinforcement learning techniques leveraging deep learning have made tremendous progress in the past years, however, their extreme complexity prevents practitioners from understanding their behavior. Decision trees have gained increased attention in supervised learning for their inherent interpretability, enabling modelers to understand the exact prediction process after learning. This paper considers the problem of optimizing interpretable decision tree policies in reinforcement learning settings. Previous works have relaxed the tree structure, restricted to optimizing only tree leaves, or applied imitation learning techniques to approximately copy the behavior of a neural network policy with a decision tree. We propose an algorithm that directly optimizes the complete decision tree using policy gradients. Our technique uses established decision tree heuristics for regression to perform proximal policy optimization, a successful policy gradient method for neural networks. Although our proposed technique tends to converge to a local optimum, we empirically show that iterating over several random seeds leads to a competitive algorithm for optimizing decision tree policies in reinforcement learning.

10.1 Introduction

In recent years many successful neural network-based techniques have been proposed for reinforcement learning. However, due to the size and structure of these models, the resulting policies cannot be precisely interpreted which limits their use in real-life applications. Decision trees are a popular type of model in supervised learning as they can be directly interpreted [1] and efficiently learned with heuristics [2, 3]. Using decision trees as reinforcement learning policies is therefore a promising research direction, but unfortunately, decision trees are difficult to optimize for reinforcement learning because they are not differentiable.

Some methods have been proposed for optimizing decision tree policies by working around their non-differentiability but they come with their caveats. VIPER [4] first trains a Deep Q-Network [5] and then extracts a decision tree from the neural network using imitation learning. While this often results in performant decision trees, it is time-consuming and relies on a good teacher model. Iterative bounding MDPs [6] and OMDT [7] require knowledge of the complete Markov Decision Process underlying the environment which means that they cannot solve most reinforcement learning problems. In another line of work, the non-differentiable parts of decision tree models have been relaxed [8, 9] or restricted [10] to enable gradient-based optimization. These relaxations or restrictions are later removed to obtain a crisp decision tree, which can perform much worse than its soft (non-crisp) counterpart. Existing methods that learn trees with a crisp structure [11, 12] do so using a greedy splitting procedure that can never update the tree structure again once creating a branch node, making it hard to recover from mistakes.

We propose the method DT-PPO which directly optimizes complete decision tree policies using Proximal Policy Optimization (PPO), a high-level overview of the method is given in Figure 10.1. PPO is a policy gradient style method that has gained widespread attention for its strong performance combined with its relative simplicity. We formulate an iterative method based on regression tree learning heuristics that can incrementally improve decision trees for a differentiable loss function. This method then allows us to use the gradient-based PPO algorithm for optimizing decision tree policies without imitation learning or altering the model class. We evaluate DT-PPO on several control tasks and discrete MDPs and compare the performance to the state-of-the-art method VIPER. DT-PPO with three random restarts performs competitively with VIPER and sometimes outperforms a neural network-based method. To the best of our knowledge, our proposed method is the first that is capable of optimizing arbitrary differentiable loss functions for decision trees. The extensible DT-PPO algorithm sets a foundation for future work in decision tree policy optimization.

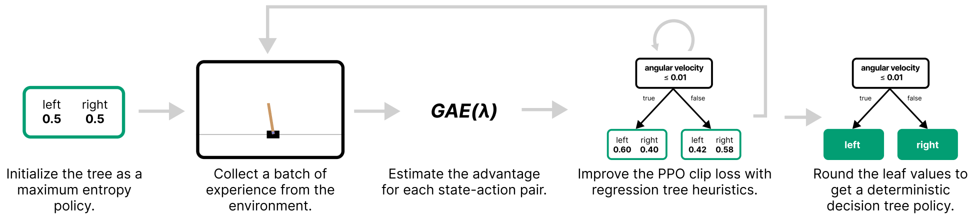

left 0.5
right 0.5
Initialize the tree as a maximum entropy policy.
Collect a batch of experience from the environment.
GAE(λ)
Estimate the advantage for each state-action pair.
angular velocity ≤ 0.01
true
false
left 0.60 right 0.40
left 0.42 right 0.58
Improve the PPO clip loss with regression tree heuristics.
angular velocity ≤ 0.01
true
false
left
right
Round the leaf values to get a deterministic decision tree policy.

Figure 10.1: High-level overview of the DT-PPO algorithm. The tree is initialized as a single leaf with equal probability for each action and iteratively refined by optimizing the Proximal Policy Optimization clip loss with regression tree heuristics on batches of environment experience. In the end, we round the leaf values to obtain an interpretable deterministic policy.

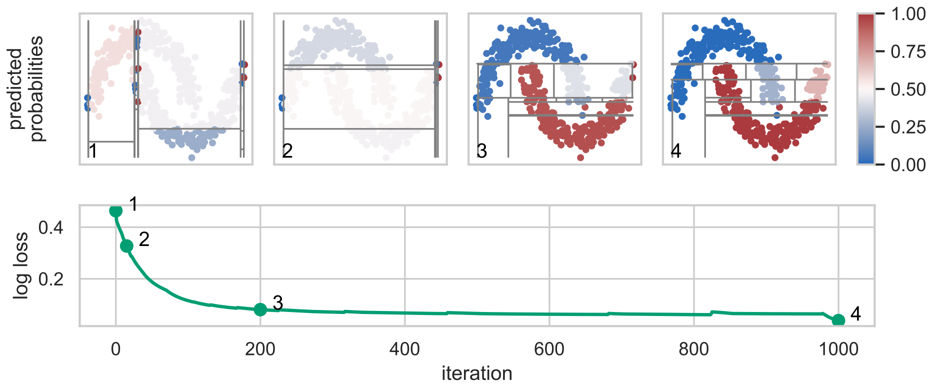


Figure 10.2: Incremental regression tree improvement for depth 5 trees on a classification task with 1000 samples, initialized with large random targets ($\mathcal{N}(0, 25)$). By applying gradient updates to the targets, we can optimize differentiable losses such as the log loss for classification.

Algorithm 8 DT-PPO: Decision Tree Proximal Policy Optimization

```
1: Initialize $\pi$ as a single leaf that takes any of the $n$ actions with equal probability: $\pi(s) = \left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right), \forall s$
2: while not reached stopping criterion do
3:     Run policy $\pi$ in the environment for $t = 1 \ldots T$ timesteps collecting observations $s_t$, actions $a_t$, and rewards $R_t$
4:     Compute the $GAE(\lambda)$ advantage estimates $\hat{A}_t = R_t(\lambda) - V_\theta(s_t)$ using lambda returns and value function network
5:     if 10th policy iteration then
6:         $\pi_0 = \pi$
7:         for $i = 1, 2, \ldots N$ do
8:             Define at time $t$ the log action probabilities as $l_t = \log(\pi_{i-1}(s_t) + 10^{-8})$
9:             Define $L^{\mathrm{DT}}(l) = \mathbb{E}_t\left[\min(r_t \hat{A}_t, \mathrm{clip}(r_t, 1-\epsilon, 1+\epsilon)\hat{A}_t) + cH(\sigma(l_t))\right],\ r_t = \frac{\sigma(l_t)_{a_t}}{\pi(a_t|s_t)}$ and $\sigma$ the softmax function
10:            Fit regression tree $\pi_i$ on observations $s_t$ with targets $Y_t = \sigma\left(l_t + \eta \nabla L^{\mathrm{DT}}(l_t)\right)$
11:        end for
12:        Choose $\pi$ as the $\pi_i$ with largest batch $L^{\mathrm{DT}}$ value
13:    else
14:        Optimize $L^{\mathrm{DT}}$ with respect to decision tree leaf values, for $N$ steps of gradient ascent
15:    end if
16:    Optimize $L^{\mathrm{VALUE}}$ with respect to neural network parameters $\theta$, for $K$ batches of size $B$ using Adam
17: end while
18: Make the policy $\pi$ deterministic by rounding each leaf's value to its argmax
```

10.2 DT-PPO: Decision Tree Proximal Policy Optimization

In this section, we describe DT-PPO: an algorithm for optimizing decision tree policies in reinforcement learning settings. DT-PPO is based on Proximal Policy Optimization, a popular policy gradient-based technique for optimizing reinforcement learning policies represented by neural networks. Previous works have usually not applied gradient-based algorithms to decision trees as decision trees are not differentiable. We propose a method that makes incremental improvements to a decision tree policy without requiring the model to be differentiable, allowing us to use the policy gradient information during optimization. We first explain our technique for incremental updates using regression tree learning heuristics followed by an explanation of how this technique is used inside the PPO algorithm.

10.2.1 Incremental Regression Tree Improvement

A major challenge in decision tree optimization is that each decision node splits on a single feature and uses a hard threshold that sends samples to the left or right subtree. This property makes the prediction space discontinuous and also has the effect that changing a single decision node can lead to a very large change in predictions. Due to these issues, existing algorithms for incremental decision tree-based learning usually resort to ensembles which comes at the cost of interpretability. For example, gradient boosting [13, 14] gradually reduces the loss by iteratively adding decision trees to its ensemble. These trees compensate for the loss of the preceding trees in the ensemble by predicting pseudo-residuals based on the gradient of the loss. We take inspiration from gradient boosting and propose an effective yet surprisingly simple method that optimizes a single decision tree for a differentiable loss function.

The main idea is to leverage regression tree learning heuristics to iteratively optimize regression trees on a sufficiently large batch of samples X for a differentiable loss function L. By updating the targets of the learner according to the gradient of the loss in each iteration we can use gradient information while not having to differentiate through the decision tree. Define the loss as a function $L : \mathcal{Y} \to \mathbb{R}$ mapping a set of model predictions to a value to be minimized. Our method works as follows:

1. initialize a decision tree $\mathcal{T}_0 : \mathcal{X} \to \mathcal{Y}$ that maps samples $x \in X$ to (arbitrary) predictions Y_0,

2. use a regression tree learner to find a tree $\mathcal{T}_i$ aiming to predict $\mathcal{T}_i \approx Y_{i-1} - \eta \nabla L(Y_{i-1})$, with learning rate η,

3. repeat step 2 for N iterations and return the best performing tree $\mathcal{T}_{\text{argmin}_i(L(Y_i))}$.

Like gradient boosting this method iteratively reduces the pseudo-residuals but instead of adding a tree to the ensemble, the new tree predicts the aggregated prediction of the ensemble with the pseudo-residuals subtracted from the targets. While this method does not guarantee an improvement of the loss value in every step we find that it works well in practice.

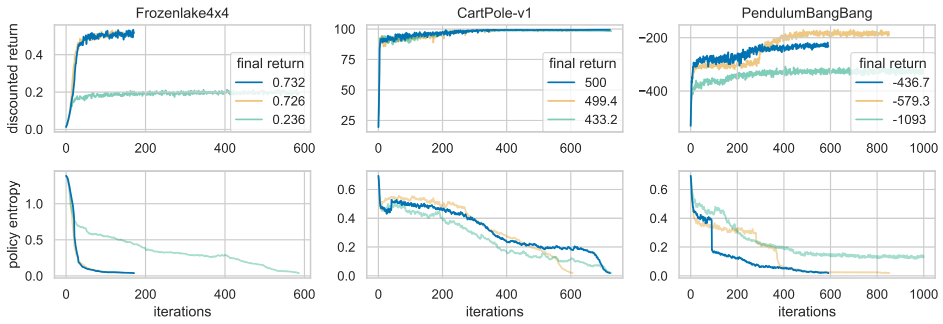


Figure 10.3: Discounted returns and policy entropy values during the DT-PPO learning process for 3 different random seeds, annotated with the final undiscounted return after rounding the non-deterministic policies. Although DT-PPO sometimes finds a bad local optimum varying the random seed helps. Entropy reduces over time with better policies converging to a deterministic policy more quickly.

We visualize the optimization behavior of incremental regression tree improvement for depth 5 trees on a simple supervised learning task in Figure 10.2. In this example, the tree is supposed to classify two moon-shaped distributions by minimizing the log loss. $\mathcal{T}_0$ is trained on large random targets sampled from a normal distribution $\mathcal{N}(0, 25)$ and then iteratively improved for 1000 iterations using regression tree learners that minimize the squared error loss. We can see that both the tree structure changes over the iterations and the leaf values are updated to predict 0 or 1 with increased confidence. The final model $\mathcal{T}_{1000}$ accurately classifies the dataset.

10.2.2 Proximal Policy Optimization

To perform Proximal Policy Optimization with decision trees we replace the neural network policy with a multi-output regression tree that predicts a probability for each action in each leaf. We then perform gradient updates with the method outlined in the previous section. The algorithm's pseudocode is given in Algorithm 8. An important requirement for this method is that the batch size is large enough to capture sufficient information about the tree of the previous iteration to not forget too much about the new tree. This is in contrast to neural network optimization where information of previous batches is contained in the model parameters and approximately persisted by limiting the magnitude of parameter updates. Therefore in our experiments, we collect batches of $T = 50000$ steps of experience similar to methods such as TRPO [15]. This number may be reduced when training smaller trees.

Aside from the batch size we make some other changes to the PPO algorithm to improve learning using decision trees. Instead of applying the incremental regression tree improvement procedure in every iteration we only perform this operation once every 10 steps. During the iterations in between we only update leaf values. We noticed that this offers a speedup as it is not necessary for the tree structure to change at every step. In DT-PPO we train regression trees that directly predict probabilities and we use softmax functions to make sure that these probabilities sum to one after gradient updates. It is also possible to predict logits with the regression trees and only use a softmax function when computing probabilities. We chose against this approach as predicting logits can result in multiple leaves with different logit values representing the same probabilities since the softmax function is invariant to constant shifts.

As mentioned in the previous section, incremental regression tree improvement does not guarantee an improvement therefore we check the loss values before and after the procedure, and if loss does not improve we instead only update leaf values. We also only keep the tree with the best batch loss after tree improvement and after terminating the whole PPO algorithm. In the rest of this paper, we limit the number of iterations to 1000 and stop the optimization process early when the mean policy entropy reaches 0.01 times the number of actions. The remaining tree learning hyperparameters are as follows: $\eta = 0.1, \epsilon = 0.2, \gamma = 0.99, \lambda = 0.95, N = 100$. To optimize decision trees with PPO, we replace the policy neural network with a decision tree but still use a neural network for the value function (the critic), as this approximator is only used to improve the optimization process and does not affect the interpretability of the policy model. We update the parameters θ of the value function in every iteration. This update minimizes PPO's clipped value loss with the Adam optimizer for 50 batches of 200 samples:

$$L^{\text{VALUE}}(\theta) = \max\left((V_\theta(s_t) - V_t)^2,\ V_{\theta_{\text{old}}(s_t)} + \text{clip}(V_\theta(s_t) - V_{\theta_{\text{old}}}(s_t), -\epsilon, \epsilon) - V_t)^2\right), \tag{10.1}$$

where V_t is the return at timestep t. For our experiments, we use a standard neural network with 2 layers of 64 neurons, tanh activation functions, and a learning rate of 2.5e-4.

Controlling Policy Entropy

Since we want to find a deterministic policy, as this improves interpretability, we need to carefully control the entropy of the policy during the learning procedure. When entropy remains too high at the end of training, the quality of the policy can drop tremendously when the probabilities in the leaves are rounded to make the policy deterministic. However, when entropy becomes low too quickly, the agent does not explore enough since PPO's exploration relies on the policy's randomness. To control the policy entropy we add the term $cH(\pi(s_t))$ (or $cH(\sigma(l_t))$ in terms of logits l_t) to the objective just like in regular PPO, but instead of fixing c to a value such as 0.01 we linearly reduce it over time. We initialize $c = 0.01$ in our experiments and reduce it by 0.01 every 100 iterations.

The final algorithm DT-PPO is still similar to PPO and differs mostly in the following ways:

- It uses a decision tree as the policy instead of a neural network, and updates it with incremental regression tree improvement (every 10th iteration).
- It uses larger batches to avoid losing information in the incremental regression tree improvement step.
- It reduces the entropy parameter c over time to reach a deterministic final policy.

			VIPER		DT-PPO		DQN	PPO
environment	feat.	act.	returns	size	returns	size	returns	
Control tasks								
Acrobot-v1	6	3	**-87.09** ± 0.35	11	-87.57 ± 1.22	3	**-65.80** ± 0.81	-78.81 ± 0.73
CartPole-v1	4	2	400.06 ± 1.57	10	**500.00** ± 0.00	4	467.65 ± 0.00	**500.00** ± 3.49
CartPoleSwingup	5	2	**466.10** ± 7.06	13	272.65 ± 7.15	12	**826.59** ± 14.87	393.83 ± 6.27
Pendulum-v1 (discrete)	3	2	**-158.48** ± 2.57	12	-436.74 ± 3.29	10	**-146.04** ± 2.70	-147.19 ± 2.87
Discrete OMDT environments								
Blackjack	3	2	**-21.68** ± 0.50	9	-23.13 ± 0.50	5	**-21.63** ± 0.52	-22.71 ± 0.51
Frozenlake4x4	2	4	**0.74** ± 0.01	7	0.73 ± 0.01	8	**0.75** ± 0.02	0.60 ± 0.01
Frozenlake8x8	2	4	0.87 ± 0.01	5	**0.90** ± 0.01	7	**0.87** ± 0.01	0.84 ± 0.01
Frozenlake12x12	2	4	**0.36** ± 0.01	9	0.05 ± 0.01	11	**0.67** ± 0.00	0.00 ± 0.02
InventoryManagement	1	100	**10278.00** ± 6.09	3	9523.55 ± 3.22	2	**10269.37** ± 5.96	10121.69 ± 5.83
Navigation3D	3	6	0.00 ± 0.00	1	**54.37** ± 0.11	5	0.00 ± 0.12	**52.36** ± 0.00
SystemAdministrator1	8	9	**1707.47** ± 3.34	10	1655.40 ± 3.37	7	**1709.90** ± 3.34	1678.94 ± 3.34
SystemAdministrator2	8	9	**1641.34** ± 6.03	6	1558.40 ± 2.83	8	**1848.41** ± 4.81	1771.05 ± 3.19
SystemAdministratorTree	7	8	**4736.65** ± 3.45	6	4373.99 ± 3.18	8	5290.16 ± 3.36	**5525.50** ± 3.23
TictactoeVsRandom	27	9	**0.71** ± 0.00	13	**0.71** ± 0.02	7	**0.99** ± 0.01	0.97 ± 0.02
TigerVsAntelope	4	5	0.38 ± 0.02	9	**1.00** ± 0.00	12	0.62 ± 0.00	**1.00** ± 0.02
TrafficIntersection	4	2	**28.07** ± 1.33	10	9.98 ± 1.43	8	**31.03** ± 1.64	-35.53 ± 1.35
Xor	2	2	508.88 ± 0.00	7	**1000.00** ± 0.00	5	**1000.00** ± 0.83	519.15 ± 0.83

Table 10.1: Mean and standard errors of undiscounted returns computed with 1000 episodes. For each method, the final policy was selected as the best of 3 random seeds. VIPER often performs somewhat better than DT-PPO but in other environments, DT-PPO improves performance as VIPER relies on the DQN to get good performance. In some environments, DT-PPO and VIPER outperform neural network policies.

10.3 Results

We compare DT-PPO's ability to optimize interpretable tree policies with VIPER [4], a method based on imitating a Q-learning teacher such as a Deep Q-Network [5]. To maintain interpretability we limit the trees to a depth of 4 and to understand the cost of interpretability we also compare against neural network-based PPO [16] and DQN. We implemented DT-PPO in JAX [17] and the reinforcement learning environments in Gymnax [18]. See our code[1] and appendix[2] for more details.

10.3.1 Learning Behavior

First, to understand the learning behavior of DT-PPO we visualize the batch discounted returns and policy entropy during optimization for Frozenlake, CartPole, and Pendulum in Figure 10.3. Optimization was stopped after 1000 iterations or when the entropy was close to zero and the best run was highlighted. DT-PPO converges to a poor local optimum in one of the Frozenlake and Pendulum runs (green). To resolve this problem we run DT-PPO three times and keep the best model after learning.

Since we are interested in finding deterministic policies we round the leaf probabilities after learning, which can also affect policy quality. For example, in the Pendulum environment, the blue run achieves a better score than the orange run after rounding while the visualized returns of the non-deterministic policies showed the opposite behavior. This is a property resulting from the simplicity of heavily regularized models such as decision trees. By using a model class that does not have the expressivity to use the complete information from the observations, an MDP implicitly becomes a partially observable MDP (POMDP). While in MDPs there is always an optimal deterministic policy [19], there are POMDPs for which the optimal policies are non-deterministic, therefore, in general, we expect to see a reduction in performance when rounding non-deterministic interpretable policies. In general, we find that most training runs converge to a low entropy and that strong policies usually converge faster.

10.3.2 Performance Comparison

Next, we compare the algorithms on a variety of environments including the MDPs implemented by OMDT [7], a set of gymnax environments including control tasks, and cartpole swingup [20]: a more challenging cartpole variant. All environments use discrete action spaces. OMDT cannot handle environments with continuous observations and unknown or large MDPs, we therefore did not directly compare to this method. Since our primary focus is on policy quality and not sample efficiency we train the neural network methods DQN and PPO for 10 million total timesteps. VIPER uses the DQN to extract a decision tree using additional environment rollouts which, when assuming rollouts of 1000 steps, results in VIPER using approximately 4.7 million additional environment samples. DT-PPO uses batches of 50,000 samples and therefore 5 million samples for each 100 iterations. In Table 10.1 we list the mean returns that the different methods achieved in the environments.

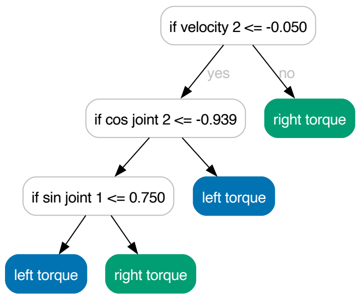
[1]`https://github.com/tudelft-cda-lab/DTPO`

[2]`https://arxiv.org/abs/2408.11632`

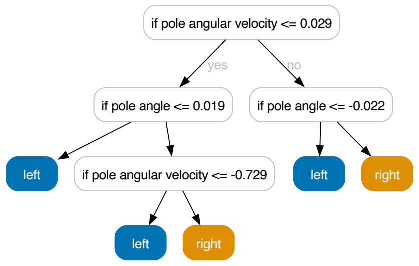
When VIPER manages to find a good policy for an environment it often performs better than DT-PPO but the scores are usually close. Since VIPER relies on the DQN to optimize its policy, the method fails in environments such as Navigation3D and TigerVsAntelope where DQN converges to a suboptimal policy. Q-learning and policy gradient techniques often excel in different types of environments as can be seen in the results of the neural network policies. Therefore DT-PPO can be useful when Q-learning techniques fail to achieve good performance.

Learned Policies

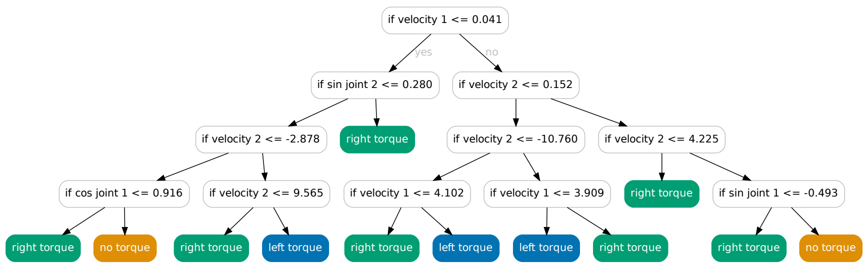
To further understand the difference between VIPER and DT-PPO we visualized the best learned trees for Acrobot-v1 and CartPole-v1 in Figure 10.4. Although all four trees perform well in the environments, the trees trained with DT-PPO use noticeably different features, and in these environments, the DT-PPO trees are smaller. We find that DT-PPO trees often contain many leaves with the same action prediction at the end of training. Such leaves are pruned at the final rounding step to produce a smaller tree. Even though all trees were allowed to train to a depth of 4, permitting a size of $2^4 - 1 = 15$ nodes, Table 10.1 shows that many trees are significantly smaller after pruning. While this results in trees that are easily interpretable it means that VIPER sometimes makes better use of its allowed capacity to achieve a higher return. In environments where this happens, users can allow a larger tree depth for DT-PPO and rely on pruning to provide sufficient size reduction for interpretability.



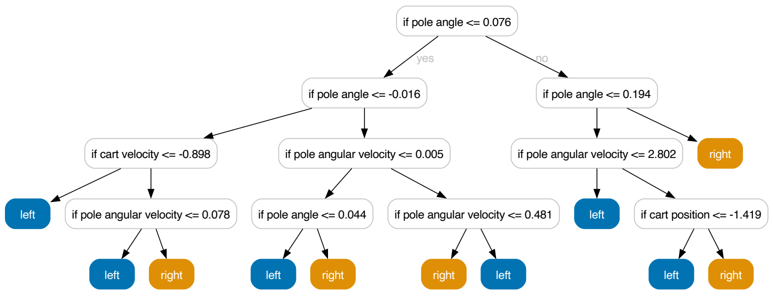
(a) DT-PPO Acrobot: -87.57



(b) DT-PPO CartPole: 500.00



(c) VIPER Acrobot: -87.09


if pole angle <= 0.076
yes
no
if pole angle <= -0.016
if pole angle <= 0.194
if cart velocity <= -0.898
if pole angular velocity <= 0.005
if pole angular velocity <= 2.802
right
left
if pole angular velocity <= 0.078
if pole angle <= 0.044
if pole angular velocity <= 0.481
left
if cart position <= -1.419
left
right
left
right
right
left
left
right

(d) VIPER CartPole: 400.06

Figure 10.4: Final decision trees optimized with DT-PPO and VIPER and their undiscounted returns. While the resulting trees score similarly they can be quite different: the trees use different features to arrive at their predictions. The DT-PPO trees converged to multiple leaves containing the same prediction which allowed for many of them to be pruned away after training.

10.4 Discussion

We presented DT-PPO, a method that can directly optimize decision tree policies for reinforcement learning. By leveraging a method based on regression tree learning heuristics to update trees with gradient information we were able to apply proximal policy optimization to decision trees. The resulting policies are small enough to be human-interpreted and performed competitively compared to existing algorithms that extract decision trees from neural network policies. Our experiments on classic control tasks and discrete MDPs demonstrate that small decision trees can sometimes perform as well as neural networks.

Although DT-PPO performed well in our benchmark there are still limitations to be addressed. While neural networks can efficiently incrementally learn from batches of data by performing weight updates with a small learning rate, our decision trees require a large batch size to function. This is because we re-learn a tree in each iteration which means that, to not forget previous experience, the batch of experience must be large enough to hold the information of the previous tree. We noticed that our method worked well for batches of 50000 samples which we efficiently collect using optimized Gymnax environments but which can be problematic in situations where sample efficiency is important. Like most function approximation-based algorithms for reinforcement learning, DT-PPO finds a local optimum. Due to the limited capacity of decision trees, these local optima sometimes score significantly worse than the global optimum which means DT-PPO requires multiple restarts to reliably find a strong policy. Mechanisms that memorize and replay previous experience could help to improve DT-PPO's sample efficiency and by memorizing multiple policies the reliance on random restarts could be reduced.

Our method provides a promising way to use gradient information inside of non-differentiable learners such as decision trees. Future work might apply this idea to different loss functions in reinforcement or supervised learning that were previously hard to optimize. We also aim to leverage the fact that policy gradient techniques such as DT-PPO can be adapted for environments with continuous action spaces with relative ease (compared to Q-learning algorithms). Lastly, recent works have proposed efficient methods for decision tree controller verification. We plan to use DT-PPO to train decision tree policies for new tasks and verify their safety properties for controllers that are currently optimized as hard-to-verify neural networks.

Bibliography

[1] Z. C. Lipton, *The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery.* Queue **16**, 31 (2018).

[2] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees* (Taylor & Francis, 1984).

[3] J. R. Quinlan, *Induction of decision trees,* Machine learning **1**, 81 (1986).

[4] O. Bastani, Y. Pu, and A. Solar-Lezama, *Verifiable reinforcement learning via policy extraction,* Advances in neural information processing systems **31** (2018).

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, *Human-level control through deep reinforcement learning,* nature **518**, 529 (2015).

[6] N. Topin, S. Milani, F. Fang, and M. Veloso, *Iterative bounding mdps: Learning interpretable policies via non-interpretable methods,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35 (2021) pp. 9923–9931.

[7] D. Vos and S. Verwer, *Optimal decision tree policies for markov decision processes,* in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, edited by E. Elkind (International Joint Conferences on Artificial Intelligence Organization, 2023) pp. 5457–5465, main Track.

[8] A. Silva, M. Gombolay, T. Killian, I. Jimenez, and S.-H. Son, *Optimization methods for interpretable differentiable decision trees applied to reinforcement learning,* in *International conference on artificial intelligence and statistics* (PMLR, 2020) pp. 1855–1865.

[9] R. R. Paleja, Y. Niu, A. Silva, C. Ritchie, S. Choi, and M. C. Gombolay, *Learning interpretable, high-performing policies for autonomous driving,* Robotics: Science and Systems XVIII (2022).

[10] A. Likmeta, A. M. Metelli, A. Tirinzoni, R. Giol, M. Restelli, and D. Romano, *Combining reinforcement learning with rule-based controllers for transparent and general decision-making in autonomous driving,* Robotics and Autonomous Systems **131**, 103568 (2020).

[11] U. D. Gupta, E. Talvitie, and M. Bowling, *Policy tree: Adaptive representation for policy gradient,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29 (2015).

[12] A. M. Roth, N. Topin, P. Jamshidi, and M. Veloso, *Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy,* arXiv preprint arXiv:1907.01180 (2019).

[13] L. Mason, J. Baxter, P. Bartlett, and M. Frean, *Boosting algorithms as gradient descent,* Advances in neural information processing systems **12** (1999).

[14] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, R. Tibshirani, and J. Friedman, *Boosting and additive trees,* The elements of statistical learning: data mining, inference, and prediction , 337 (2009).

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, *Trust region policy optimization,* in *International conference on machine learning* (PMLR, 2015) pp. 1889–1897.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms,* arXiv preprint arXiv:1707.06347 (2017).

[17] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: composable transformations of Python+NumPy programs,* (2018).

[18] R. T. Lange, *gymnax: A JAX-based reinforcement learning environment library,* (2022).

[19] M. L. Puterman, *Markov decision processes,* Wiley Series in Probability and Statistics (1994).

[20] D. Ha, *Evolving stable strategies,* blog.otoro.net (2017).

11 Discussion

In this dissertation, we considered the problem of optimizing decision trees for hard-to-optimize tasks. To answer the question 'How to learn robust decision trees?' in Part I, we developed algorithms for fast and optimal robust optimization and also worked on decision trees robust to data poisoning. In Part II we answered the question 'How to learn decision trees as policies for sequential decision problems' by proposing an exact algorithm for decision tree policy optimization in Markov Decision Processes, and developed a reinforcement learning method for decision trees. We will not repeat the conclusions to the research questions as these are contained in each chapter. Instead, what follows is a list of unexpected results, a discussion of general considerations and a statement on the limitations of this dissertation.

11.1 Unexpected Results

- Early comparisons (e.g. Chapter 4) of tree ensemble adversarial robustness showed that many robust ensembles performed similarly. However, it turns out that many comparisons were evaluated with a perturbation size that was either too small, resulting in all models trivially providing robustness, or too large, resulting in models not being able to perform better than random guessing. There are clear differences when choosing the perturbation size carefully to disallow trivial solutions (Chapter 5).

- The adversarial robustness of gradient boosting ensembles can typically be verified orders of magnitude faster than the robustness of random forests when the ensembles have the same number of trees. This is because random forests train deep decision trees that continue splitting until no further splits improve performance, while gradient boosted trees are typically limited to a small size. Although the ensembles have the same number of trees, the number of total leaf nodes differs by orders of magnitude.

- Simply training regular random forests and relabeling their trees for robustness leads to similar robustness as the specialized method GROOT. This is especially useful as robust relabeling can be easily done for perturbations other than those bounded by an L^∞ ball of fixed radius.

- In rare occasions, training differentially private decision trees, on average, improved test accuracy over non-private decision trees. This is because adding a small amount of noise can produce a helpful regularization effect, but is counter intuitive as there is typically a clear trade-off between privacy and performance.
- Simple (small) deterministic decision tree policies can be remarkably effective and still produce non-trivial behavior by using the environment's non-determinism. For instance, in the Frozenlake game (Chapter 9), decision tree policies often run the agent directly into a wall to use the randomness of the environment to move the agent along that wall.
- By performing gradient updates only to determine the desired outputs of a decision tree and training with regression tree heuristics, it is possible to optimize decision trees using gradient-based methods like Proximal Policy Optimization. Large batches of data can hold enough information on the previous policy to prevent excessive changes without relying on making small changes to the model parameters.

11.2 Considerations and Takeaways

There are some questions and problems that regularly arise when optimizing decision trees. We discuss the important considerations below.

Optimal or Heuristic In this dissertation, we have proposed both optimal and heuristic methods for learning decision trees. The advantages of optimal methods are two-fold: they perform as well or better than heuristic methods and are provably optimal, i.e., they prove no performance improvement can be made. However, optimal methods often take orders of magnitude longer to run, and their time complexities scale exponentially, whereas heuristics often scale polynomially. This means that in practice, one must consider whether it is worth it to pay the price in runtime for the limited performance gain that optimal methods achieve and this will depend on the use case.

A common misconception is treating the optimality guarantee of optimal decision trees to mean optimal at testing time. It is important to interpret the guarantee correctly: given a model size constraint and the specified feature set, the optimal decision tree achieves the best possible loss on the training data. This means that test scores of other decision trees can actually be better than the optimal tree, for instance, when the size of the decision tree is increased, when new engineered features are included, or when the test set is generalized to better. This should be taken into account when applying optimal decision trees in practice.

Robust or Non-Robust Part I of this dissertation studies the problem of optimizing robust decision trees. The most common algorithms that are used in practice do not take robustness into account which makes them susceptible to small changes in the dataset. This can be problematic when data is collected from noisy samples for example. Particularly, one should consider whether deployed machine learning models are susceptible to malicious user inputs since such adversarial examples can be directly prevented with robust models. While robustness can be an important property and can be used as a tool to prevent overfitting, there are costs associated with robust optimization: the resulting solutions often sacrifice some predictive performance for robustness and the models take longer to train. In practice, it should be considered whether the extra costs of robust optimization are worth it.

Transparency or Performance Single decision tree models are popular due to their interpretability and verifiability properties, but there is a trade-off between transparency and performance. Some use cases require more expressive models than size-limited decision trees to achieve acceptable performance. For example, the language modeling tasks performed by programs such as ChatGPT are naturally more difficult and require more complex models than those required for balancing a pole on a cart (Chapter 10). Therefore when necessary, one should resort to models like neural networks to trade transparency for performance. However, much of the machine learning literature exclusively considers opaque neural networks and never investigates whether a more transparent model can also solve the task. When applying machine learning, one should consider evaluating various models and choose the most transparent model that performs satisfactorily.

Defining Uncertainty Every robust optimization problem requires a definition of the uncertainty set. Unfortunately, it can be difficult in practice to define the right uncertainty set to robustify against. In some situations, one can make reasonable estimates; for example, if a feature comes from a sensor value with a known uncertainty, one might use that value, or when robustness is used as a regularization tool, the uncertainty size can be tuned based on a validation set. However, in other situations, it can be impossible to know the exact uncertainty. For example, it can be unknown by what extent features are likely to change. Whenever robust optimization is applied in practice, one must consider how the uncertainty set can be reasonably defined.

Reinforcement Learning or Incorporating Knowledge In the field of reinforcement learning, the default assumption is that information such as action and observation spaces are known, but not the transition probabilities and the reward function. However, when formulating reinforcement learning problems, we often have intricate knowledge of the environments since we typically implement them as interactive simulators ourselves. This brings up the question of whether we should also include this kind of information in the learning algorithms. For example, in Chapter 9, solving the tic-tac-toe problem the way it was given required a deep decision tree. However, if one included their knowledge of rotational symmetries in the environment, the game would be easier to solve and would require a smaller decision tree. Similarly, superhuman reinforcement learning chess bots rely on Monte Carlo Tree Search which uses knowledge of the chess rules. Without this search step, the model is still good (about 2500 ELO) but not superhuman[1]. In practice, one should consider including knowledge about the problem in their learning methods instead of applying pure reinforcement learning where the environment is assumed to be purely an unknown oracle.

[1]`https://lichess.org/@/LazyBot`

Numerical or Non-Numerical Data Many machine learning algorithms are designed to work on inputs that are represented by a fixed-size vector with numerical values. In such vector spaces, the algorithms often rely on distance norms computed between the input vectors as a proxy for (dis)similarity. For example, an imperceptible change for an adversarial example is typically defined by a ball with a small radius placed around the original data point. The problem with these approaches is that they do not work when the data is not numerical.

Categorical features are common in tabular data, which is often used for decision trees. Distances computed on naive encodings of categorical features do not provide valid distances. Even when working only with numerical data, features typically have different data scales, preventing useful distance computation. Even worse, when training on identically scaled numerical features in high dimensions, intuitions about distance and convexity also no longer apply. For example, the probability for a data point to lie within the convex hull of previous data points approaches zero when the dimension increases. Therefore, it is important to consider the heterogeneity in feature types and lack of useful distances when designing algorithms for real-world tabular data problems.

11.3 Limitations

The conclusions that can be drawn from the research in this dissertation are limited in some cases. We discuss these limitations below.

Testing on Public Benchmark Datasets Machine learning is always based on data. Unfortunately, datasets often contain private information or hold competitive value, which makes organizations hesitant to share them. This means that in our experimentation, we were limited in the kinds of data on which we could test our proposed methods. Throughout this dissertation, we have used publicly available benchmarks from the UCI machine learning repository and OpenML. Therefore it is possible that not all results extend to private datasets with different properties.

Unrealistic Uncertainty Sets As discussed before, choosing the correct uncertainty set to robustify against is a very difficult problem. In Chapter 4, we used a fixed perturbation radius for every dataset, some of which turned out to be too large and, therefore, led to trivial models that predict the same class all the time being learned. In Chapters 5 and 6, we used a bound on adversarial accuracy to choose 3 different perturbation sizes per dataset. These perturbation sizes led to the algorithms learning non-trivial models but are still not necessarily realistic. Therefore it is unclear how well our results generalize to uncertainty sets for real use cases.

Simple Markov Decision Processes When we evaluated decision tree policies in Part II, we did so on two sets of benchmarks: control tasks from the Gymnasium library and discrete MDPs that we implemented based on the literature. Both of these benchmarks contained relatively simple tasks, meaning that small decision trees were sufficient to perform well in the majority of the environments. There are environments outside of our evaluation that are, by design, too complex to solve with small decision trees. For example, a problem that requires 20 different actions to be taken for a good policy requires a decision tree with at least 20 leaf nodes. Therefore our results do not necessarily extend to larger, more complex environments.

Size-Constrained Decision Trees Throughout this dissertation, we have considered single decision tree models with the goal of interpretability in mind. This means that the decision trees were limited to a depth of 3, 4, or 5 and not trained to be any larger. This enables humans to interpret the models as they consist of at most 8, 16, or 32 leaf nodes. In some use cases, however, a decision tree is not required to be directly human-interpretable, but it is sufficient for the policy to be machine-verifiable. In practice this means that one might want to apply our algorithms to train trees of significantly larger depths in those cases. We have not evaluated the performance of decision trees in this range.

Predicate Types In this dissertation, we have limited ourselves to binary decision trees with mainly one kind of predicate: 'feature value $\leq$ threshold.' We have also used separate kinds of predicates for categorical features. In practice, it can be useful to use non-binary decision trees with categorical variables, e.g., by splitting a feature with 3 possible values into 3 subtrees. While different kinds of predicates can be encoded into binary datasets through pre-processing and natively learned with our proposed approaches, we cannot directly learn non-binary decision trees.

11.4 Future Work

Robustness to Uncertainty Sets Other Than L^∞ Balls While plenty of work has been done on learning robust decision trees against L^∞ norm-bounded perturbations, few methods exist for other perturbation sets. Our work on robust relabeling provides a general method that works for other distance norms, but it does not optimize the complete tree. Future work might develop algorithms that flexibly learn robust decision trees against various perturbation sets. Moreover, the uncertainty sets considered in the field of adversarial examples are pessimistic and assume every sample can be changed independently. More research into methods for training distributionally-robust decision trees can prove useful in settings where perturbations behave more stochastic than adversarial. Other research could study how to specify the uncertainty set, for example, by using data-driven approaches for estimating uncertainty.

Scaling Optimal Trees Learning optimal decision trees is notoriously slow. Optimal decision trees for classification have been sped up tremendously using, for instance, dynamic programming techniques. Such techniques cannot directly be applied to robust optimization and sequential decision-making problems however. Future work may look into adaptions of dynamic programming techniques to train optimal decision trees for robustness and sequential decision-making. Another approach would be to improve the time for good sub-optimal solutions to be returned by integer programming solvers, for example by including better heuristics for decision tree learning within the search algorithm.

Optimal Differentially-Private Trees Differential privacy is a promising technique for protecting user privacy as it provides guarantees that hold even after post-processing. Progress in the field has resulted in methods that achieve the best possible solution on average for any specified level of privacy. For example, when counting the number of elements in a set and privately releasing these counts, we know exactly how much noise to add from the Geometric distribution to satisfy the privacy guarantee while maximizing the accuracy of the noisy output. Future work might investigate methods that approach this property for decision tree learning. Current methods for differentially-private decision trees optimize trees greedily and add noise to each operation. Greedy trees do not allow for an approximation guarantee. A method akin to optimal trees in the differentially private setting can improve the practicality of decision trees in the high privacy regime.

Decision Trees Beyond Interpretability As mentioned in the limitations, our focus is on interpretable models when training single decision trees. This means that decision trees were limited in size to be easier for humans to understand. However, in some practical applications, humans do not have to understand every detail of the model and can instead use computers to verify important properties of the models. In that setting, it is possible to train larger trees that express more complex relationships and thus are more performant in complex tasks. For interpretability, we have also avoided including linear relations inside our decision trees, for example, in the way that this is done in oblique decision trees. However, we expect linear relations to be useful when expressing policies, especially in control tasks such as those found in reinforcement learning. Therefore future work can consider training deeper trees with more complex parts as long as these trees remain tractably verifiable.

Glossary

3SAT 3 Boolean Satisfiability, the NP-complete Boolean Satisfiability problem for clauses with three literals.

ASR Attack Success Rate, a metric used in data poisoning attacks that measures how often an attack such as implementing a backdoor succeeds.

BDPT Building a Differentially Private Tree, a greedy algorithm for learning differentially private decision trees.

CART Classification and Regression Trees, a popular greedy algorithm for training decision trees.

CNF Conjunctive Normal Form, a way to express Boolean Formulas as a conjunction (‘and’ operations) over disjunctions (‘or’ operations).

CP Constraint Programming, a principle for solving satisfiability problems on a set of (restrictive) constraints.

DNF Disjunctive Normal Form, a way to express Boolean Formulas as a disjunction (‘or’ operations) over conjunctions (‘and’ operations).

DP Dynamic Programming, a high-level technique for speeding up algorithms by using the ‘optimal substructure’ property that exists in certain problems.

DP Differential Privacy, a technique used to provide user data privacy by adding randomized noise inside of an algorithm.

DPA Deep Partition Aggregation, a method for defending against data poisoning attacks by creating an ensemble of models all trained on disjoint subsets of the data.

DPGDF Differentially Private Greedy Decision Forest, a greedy algorithm for learning differentially private decision trees.

DQN Deep Q-Network, a Q-learning-based approach of training neural networks for reinforcement learning.

DT Decision Tree, a hierarchical model for making predictions based on informative features.

EM Exponential Mechanism, a mechanism for differential privacy for selecting a solution with high utility from a set of candidates.

GAE Generalized Advantage Estimation, a method used in reinforcement learning to estimate how much better it was to take an action in a certain state compared to what was expected.

GBDT Gradient Boosted Decision Trees, an ensemble model combining many small regression trees into a single stronger predictor.

GROOT Growing Robust Trees, a fast method that we propose to train robust decision trees against box-shaped adversaries.

ILP Integer Linear Program, an optimization problem with a linear objective and set of linear constraints, defined for integer variables.

LLM Large Language Model, a high-level term often used for large pre-trained transformer models used for language modeling.

LP Linear Program, an optimization problem with a linear objective and set of linear constraints, defined for rational variables.

LSU Linear SAT-UNSAT, an anytime exact algorithm for solving MAX-SAT problems.

MaxSAT Maximum Boolean Satisfiability, the optimization version of SAT, find a solution that maximizes the number of satisfied clauses.

MDP Markov Decision Process, a model for sequential decision making problems.

MILP Mixed Integer Linear Program, an optimization problem with a linear objective and set of linear constraints, defined for a mix of rational and integer variables.

OMDT Optimal MDP Decision Tree, a method that we propose to train optimal decision tree policies for known discrete Markov Decision Processes.

PF Permute-and-Flip Mechanism, a mechanism for differential privacy for selecting a solution with high utility from a set of candidates.

PPO Proximal Policy Optimization, a modern policy gradient-based technique for optimizing reinforcement learning agents.

RC2 Relaxable Cardinality Constraints, an exact algorithm for solving MAX-SAT problems.

RF Random Forest, an ensemble model combining many deep decision trees into a single stronger predictor.

ROCT Robust Optimal Classification Trees, a method that we propose to train optimal robust decision trees against adversarial examples modeled with a box-shaped attacker.

SAT Boolean Satisfiability, an NP-complete problem that asks for a satisfying assignment to a Boolean formula in Conjunctive Normal Form.

TREANT Training Evasion-Aware Decision Trees, an algorithm for training robust decision trees against adversaries described by a set of perturbation rules.

UNSAT Unsatisfiable, a term often used in SAT solving to refer to a Boolean formula that is proven never to be true.

VIPER Verifiability via Iterative Policy ExtRaction, an algorithm based on imitation learning for distilling complex Q-learners into verifiable models such as decision trees.

Curriculum Vitæ

Daniël Alexander Vos

March 6, 1998	Born in Delft, The Netherlands
2009 – 2015	Gymnasium diploma Grotius College Delft
2015 – 2018	BSc. Computer Science & Engineering *(cum laude)* Delft University of Technology Honours Programme Minor Computer Science at ETH Zürich Thesis internship at ING Group, Amsterdam
2018 – 2020	MSc. Computer Science *(cum laude)* Delft University of Technology Specializing in Cyber Security and Data Science
2020 – 2024	PhD in Computer Science Delft University of Technology Cyber Security and Algorithmics research groups
Feb. – Apr. 2024	Visiting Researcher at USC, Los Angeles
2024 – present	Postdoctoral Researcher Delft University of Technology Algorithmics research group

List of Publications

🗎 1. **Daniël Vos**, and Sicco Verwer. ‘Optimal Decision Tree Policies for Markov Decision Processes’. In Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, edited by Edith Elkind, 5457–65. International Joint Conferences on Artificial Intelligence Organization, 8 2023.

2. Azqa Nadeem, **Daniël Vos**, Clinton Cao, Luca Pajola, Simon Dieck, Robert Baumgartner, and Sicco Verwer. ”Sok: Explainable machine learning for computer security applications.” In 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P), pp. 221-240. IEEE, 2023.

3. Yingqian Zhang, Laurens Bliek, Paulo da Costa, Reza Refaei Afshar, Robbert Reijnen, Tom Catshoek, **Daniël Vos** et al. ”The first AI4TSP competition: Learning to solve stochastic routing problems.” Artificial Intelligence 319 (2023): 103918.

4. Jelle Vos, **Daniël Vos**, and Zekeriya Erkin. ”Efficient Circuits for Permuting and Mapping Packed Values Across Leveled Homomorphic Ciphertexts.” In European Symposium on Research in Computer Security, pp. 408-423. Cham: Springer International Publishing, 2022.

🗎 5. **Daniël Vos**, and Sicco Verwer. ”Adversarially Robust Decision Tree Relabeling.” In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 203-218. Cham: Springer Nature Switzerland, 2022.

🗎 6. **Daniël Vos**, and Sicco Verwer. ”Robust optimal classification trees against adversarial examples.” In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, no. 8, pp. 8520-8528. 2022.

🗎 7. **Daniël Vos**, and Sicco Verwer. ”Efficient training of robust decision trees against adversarial examples.” In International Conference on Machine Learning, pp. 10586-10595. PMLR, 2021.

Under submission

🗎 8. **Daniël Vos**, Sicco Verwer. ”DT-PPO: Interpretable Proximal Policy Optimization using Decision Trees.” 2024.

🗎 9. **Daniël Vos**, Jelle Vos, Tianyu Li, Zereriya Erkin, Sicco Verwer. ”Differentially-Private Decision Trees with Probabilistic Robustness to Data Poisoning.” preprint available: arXiv:2305.15394, 2023.

🗎 Included in this dissertation.

Titles in the SIKS Dissertation Series since 2016

2016	01	Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
	02	Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
	03	Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
	04	Laurens Rietveld (VUA), Publishing and Consuming Linked Data
	05	Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
	06	Michel Wilson (TUD), Robust scheduling in an uncertain environment
	07	Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
	08	Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
	09	Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
	10	George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
	11	Anne Schuth (UvA), Search Engines that Learn from Their Users
	12	Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
	13	Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
	14	Ravi Khadka (UU), Revisiting Legacy Software System Modernization
	15	Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
	16	Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
	17	Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
	18	Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
	19	Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
	20	Daan Odijk (UvA), Context & Semantics in News & Web Search
	21	Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
	22	Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
	23	Fei Cai (UvA), Query Auto Completion in Information Retrieval

24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis

50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation

03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines

04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store

05 Mahdieh Shadi (UvA), Collaboration Behavior

06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search

07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly

08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery

09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text

10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment

12 Sander Leemans (TU/e), Robust Process Mining with Guarantees

13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology

14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior

15 Peter Berck (RUN), Memory-Based Text Correction

16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines

17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution

18 Ridho Reinanda (UvA), Entity Associations for Search

19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval

20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility

21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)

22 Sara Magliacane (VUA), Logics for causal inference under uncertainty

23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces

24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch

27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

28 John Klein (VUA), Architecture Practices for Complex Contexts

29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (TiU), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VUA), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations

	10	Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
	11	Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
	12	Xixi Lu (TU/e), Using behavioral context in process mining
	13	Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
	14	Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
	15	Naser Davarzani (UM), Biomarker discovery in heart failure
	16	Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
	17	Jianpeng Zhang (TU/e), On Graph Sample Clustering
	18	Henriette Nakad (UL), De Notaris en Private Rechtspraak
	19	Minh Duc Pham (VUA), Emergent relational schemas for RDF
	20	Manxia Liu (RUN), Time and Bayesian Networks
	21	Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
	22	Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
	23	Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
	24	Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
	25	Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
	26	Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
	27	Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
	28	Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
	29	Yu Gu (TiU), Emotion Recognition from Mandarin Speech
	30	Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
2019	01	Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
	02	Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
	03	Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
	04	Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
	05	Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
	06	Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
	07	Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
	08	Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes

09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems
10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VUA), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES

35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming
36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OU), Learning visually grounded and multilingual representations

---

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TU/e), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots

	25	Xin Du (TU/e), The Uncertainty in Exceptional Model Mining
	26	Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
	27	Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
	28	Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
	29	Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
	30	Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
	31	Gongjin Lan (VUA), Learning better – From Baby to Better
	32	Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
	33	Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
	34	Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
	35	Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production
2021	01	Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
	02	Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
	03	Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
	04	Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
	05	Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
	06	Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
	07	Armel Lefebvre (UU), Research data management for open science
	08	Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
	09	Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
	10	Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
	11	Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
	12	Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
	13	Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
	14	Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
	15	Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
	16	Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
	17	Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks

	18	Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
	19	Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
	20	Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
	21	Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
	22	Sihang Qiu (TUD), Conversational Crowdsourcing
	23	Hugo Manuel Proença (UL), Robust rules for prediction and description
	24	Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
	25	Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
	26	Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
	27	Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
	28	Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs
2022	01	Judith van Stegeren (UT), Flavor text generation for role-playing video games
	02	Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
	03	Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
	04	Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
	05	Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
	06	Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
	07	Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
	08	Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
	09	Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
	10	Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
	11	Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
	12	Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
	13	Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
	14	Michiel Overeem (UU), Evolution of Low-Code Platforms

	15	Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
	16	Pieter Gijsbers (TU/e), Systems for AutoML Research
	17	Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
	18	Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
	19	Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
	20	Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
	21	Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
	22	Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
	23	Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
	24	Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
	25	Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
	26	Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
	27	Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
	28	Onuralp Ulusoy (UU), Privacy in Collaborative Systems
	29	Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
	30	Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
	31	Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
	32	Cezara Pastrav (UU), Social simulation for socio-ecological systems
	33	Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
	34	Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
	35	Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction
2023	01	Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
	02	Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
	03	Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
	04	Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval

05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
22 Alireza Shojaifar (UU), Volitional Cybersecurity
23 Theo Theunissen (UU), Documentation in Continuous Software Development
24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions
28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

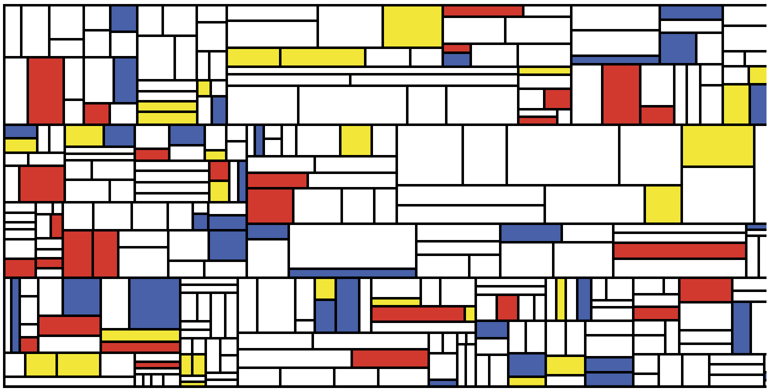
2024	01	Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
	02	Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
	03	Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
	04	Mike Huisman (UL), Understanding Deep Meta-Learning
	05	Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
	06	Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
	07	Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
	08	Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
	09	Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
	10	Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
	11	withdrawn
	12	Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
	13	Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
	14	Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
	15	Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
	16	Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
	17	Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
	18	Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
	19	Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
	20	Ritsart Anne Plantenga (UL), Omgang met Regels
	21	Federica Vinella (UU), Crowdsourcing User-Centered Teams
	22	Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
	23	Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
	24	Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
	25	Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
	26	Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
	27	Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning

28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs
29 David Wilson Romero Guzman (VUA), The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases
30 Vijanti Ramautar (UU), Model-Driven Sustainability Accounting
31 Ziyu Li (TUD), On the Utility of Metadata to Optimize Machine Learning Workflows
32 Vinicius Stein Dani (UU), The Alpha and Omega of Process Mining
33 Siddharth Mehrotra (TUD), Designing for Appropriate Trust in Human-AI interaction
34 Robert Deckers (VUA), From Smallest Software Particle to System Specification - MuDForM: Multi-Domain Formalization Method
35 Sicui Zhang (TU/e), Methods of Detecting Clinical Deviations with Process Mining: a fuzzy set approach
36 Thomas Mulder (TU/e), Optimization of Recursive Queries on Graphs
37 James Graham Nevin (UvA), The Ramifications of Data Handling for Computational Models
38 Christos Koutras (TUD), Tabular Schema Matching for Modern Settings
39 Paola Lara Machado (TU/e), The Nexus between Business Models and Operating Models: From Conceptual Understanding to Actionable Guidance
40 Montijn van de Ven (TU/e), Guiding the Definition of Key Performance Indicators for Business Models
41 Georgios Siachamis (TUD), Adaptivity for Streaming Dataflow Engines
42 Emmeke Veltmeijer (VUA), Small Groups, Big Insights: Understanding the Crowd through Expressive Subgroup Analysis
43 Cedric Waterschoot (KNAW Meertens Instituut), The Constructive Conundrum: Computational Approaches to Facilitate Constructive Commenting on Online News Platforms
44 Marcel Schmitz (OU), Towards learning analytics-supported learning design
45 Sara Salimzadeh (TUD), Living in the Age of AI: Understanding Contextual Factors that Shape Human-AI Decision-Making
46 Georgios Stathis (Leiden University), Preventing Disputes: Preventive Logic, Law & Technology
47 Daniel Daza (VUA), Exploiting Subgraphs and Attributes for Representation Learning on Knowledge Graphs
48 Ioannis Petros Samiotis (TUD), Crowd-Assisted Annotation of Classical Music Compositions

---

2025 01 Max van Haastrecht (UL), Transdisciplinary Perspectives on Validity: Bridging the Gap Between Design and Implementation for Technology-Enhanced Learning Systems
02 Jurgen van den Hoogen (JADS), Time Series Analysis Using Convolutional Neural Networks
03 Andra-Denis Ionescu (TUD), Feature Discovery for Data-Centric AI
04 Rianne Schouten (TU/e), Exceptional Model Mining for Hierarchical Data

05 Nele Albers (TUD), Psychology-Informed Reinforcement Learning for Situated Virtual Coaching in Smoking Cessation
06 Daniël Vos (TUD), Decision Tree Learning: Algorithms for Robust Prediction and Policy Optimization
07 Ricky Maulana Fajri (TU/e), Towards Safer Active Learning: Dealing with Unwanted Biases, Graph-Structured Data, Adversary, and Data Imbalance
08 Stefan Bloemheuvel (TiU), Spatio-Temporal Analysis Through Graphs: Predictive Modeling and Graph Construction



The graphic on this cover visualizes the splits of a random decision tree of depth 10, and paints its 894 leaves randomly in colors inspired by the paintings of famous Dutch painter Piet Mondriaan.
One of the most challenging aspects of decision tree optimization is identifying where to place the splits to partition the space into useful regions.
Random splits are generally not the most useful, but they are pretty!