Improving Decision Tree Learning

Peng Yu

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAT037

Thèse de doctorat

Improving Decision Tree Learning

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique, données, IA

Thèse présentée et soutenue à Montréal par visioconférence, le 2024-12-11, par

PENG YU

Composition du Jury :

Daniel Aloise Department of Computer and Software Engineering, École Polytechnique de Montréal	Président / Rapporteur
Pinghui Wang Department of Computer Science, Xi'an Jiaotong University	Rapporteur
Maroua Bahri LIP6, Sorbonne Université	Examinateur
Ciprian Daniel Neagu SFHEA Department of Computer Science, University of Bradford	Examinateur
Albert Bifet LTCI, Télécom Paris	Directeur de thèse
Jesse Read LIX, Bâtiment Alan Turing, Ecole Polytechnique	Co-directeur de thèse

To all those who helped me
to make this dream come true.
“Feeling gratitude and not expressing it is like wrapping a present
and not giving it”.
-William Arthur Ward -

Acknowledgements

Completing this doctoral journey has been a profound and transformative experience filled with challenges and triumphs. Reflecting on this remarkable path, I am deeply grateful to all those who have guided, supported, and inspired me.

First and foremost, I extend my sincere gratitude to my primary mentor, Prof. Albert Bifet. His dedicated mentorship, insightful feedback, and extensive knowledge have been instrumental in shaping the trajectory and quality of this thesis. Prof. Bifet's collaborative spirit and unwavering belief in my potential have set high standards that continuously motivated me to strive for excellence. Under his guidance, I have gained a deeper understanding of my field, developed resilience, and cultivated a passion for lifelong learning.

I am also profoundly thankful to my co-supervisor, Prof. Jesse Read, whose invaluable contributions have significantly influenced this work. Prof. Read's passion for innovative research and deep insights have been a driving force in pushing the boundaries of knowledge. His expertise in complex concepts and methodologies has greatly enriched my development as a researcher. I am fortunate to have benefited from his guidance and constant encouragement.

The academic community surrounding and nurturing me throughout this journey deserves special recognition. The exchange of ideas, intellectual debates, and collaborative opportunities have infused my research with energy and enthusiasm. Engaging with fellow researchers, participating in thought-provoking discussions, and working alongside brilliant minds have significantly enhanced the quality of this thesis. It is within this vibrant intellectual ecosystem that my academic growth has flourished.

I am deeply grateful to my colleagues, friends, and family, whose unwavering support, patience, and encouragement have been my foundation. Their belief in my potential has provided strength during challenging times, resilience in the face of setbacks, and shared joy in moments of success. This journey would have been more difficult without their steadfast support and faith in my abilities.

Lastly, I would like to express my gratitude to the individuals, organizations, and institutions contributing to this research. Their trust and collaboration have been pivotal in bridging the gap between theory and practice, enabling the realization of impactful real-world applications. Without their support, the foundations of this work would not have been as strong.

This thesis is a testament to the collective efforts of many individuals whose contributions, guidance, and support have been invaluable. Though this academic journey concludes here, the lessons learned, relationships forged, and knowledge gained will continue to resonate in my future endeavors. This thesis is not merely the result of years of research but also a reflection of the spirit of gratitude and shared wisdom that permeates the academic community. As I look ahead to new horizons, I carry with me the lessons, values, and connections cultivated during this remarkable journey, underscoring the enduring impact of mentorship, collaboration, and unwavering support in the pursuit of knowledge and excellence.

Résumé

L'apprentissage automatique (machine learning) joue un rôle central dans la science des données et l'informatique moderne, en particulier pour l'analyse de données structurées. Parmi les différentes approches disponibles, les arbres de décision suscitent un intérêt constant grâce à leur **simplicité d'interprétation** et à leurs **performances** solides dans de nombreux scénarios. Un arbre de décision peut se représenter comme un organigramme où chaque nœud correspond à une question portant sur une variable, et chaque branche mène progressivement à une prédiction finale (classe, valeur numérique, etc.). Cette structure hiérarchisée confère aux arbres de décision une lisibilité appréciable, tant pour les spécialistes que pour les acteurs métiers souhaitant comprendre les raisons sous-tendant une prédiction.

Cependant, à mesure que les arbres grandissent et se complexifient, ils deviennent plus difficiles à interpréter. Pour répondre à ce défi, des approches d'explicabilité avancées ont émergé, notamment l'utilisation des valeurs de Shapley et de l'algorithme *TreeShap*. Ces méthodes attribuent un score d'importance à chaque variable et chaque fraction de branchement, permettant ainsi d'expliquer pourquoi un arbre de décision aboutit à une certaine prédiction. Malgré leur intérêt, ces techniques exigent souvent des ressources de calcul importantes, ce qui limite leur applicabilité à grande échelle ou sur des modèles très profonds. C'est dans ce contexte que nous proposons l'algorithme **Linear TreeShap**, qui offre un compromis entre *efficacité de calcul* et *qualité d'interprétation*. L'objectif est de faciliter l'analyse des modèles d'arbres même lorsqu'ils deviennent particulièrement complexes.

En parallèle, la gestion des variables catégorielles dans les arbres de décision demeure une **problématique clé**. Contrairement aux variables numériques, que l'on peut aisément ordonner et segmenter (par exemple, séparer des âges en classes d'intervalle), les variables qualitatives—telles que les couleurs, les catégories de produits ou les types d'événements—ne possèdent pas d'ordre naturel. Lorsque le nombre de catégories devient important, le nombre de partitions possibles explose, rendant les calculs coûteux et parfois sous-optimaux. Les méthodes habituelles de conversion (par exemple, le *one-hot encoding*) peuvent fonctionner dans le cas de la classification binaire, mais elles montrent leurs limites dans des scénarios à classes multiples, comme prédire plusieurs couleurs (rouge, vert, jaune) ou différents types d'objets.

Face à ce constat, nous présentons un **cadre méthodologique** permettant aux arbres de décision—et plus particulièrement aux algorithmes de type *CART* (*Classification and Regression Trees*)—d'exploiter directement les variables qualitatives lors des opérations de scission, sans recourir systématiquement à la conversion numérique. Cette approche vise non seulement à renforcer la **précision** du modèle lorsque plusieurs classes sont en jeu, mais aussi à mieux refléter la logique intrinsèque des catégories (par exemple, le fait qu'il n'y a pas de continuum numérique entre "rouge" et "vert").

Pour répondre aux défis liés aux grands ensembles de catégories, nous introduisons également la méthode **BSplitZ**, qui repose sur une approche *stochastique* pour scinder efficacement des jeux de données contenant des variables catégorielles volumineuses. L'idée est de réduire la complexité exponentielle tout en maintenant un niveau de performance élevé. Cette méthode s'avère particulièrement adaptée aux applications à grande échelle, où le nombre de catégories distinctes peut se chiffrer en dizaines, voire en centaines.

Un autre aspect déterminant dans la construction des arbres concerne la **sélection des critères de scission**. Dans le cadre de la régression, plusieurs métriques peuvent être utilisées, parmi lesquelles l'erreur absolue moyenne (*MAE*, *Mean Absolute Error*). Cette métrique est appréciée pour sa robustesse vis-à-vis des valeurs aberrantes et des distributions fortement déséquilibrées. Toutefois, l'application de la MAE aux variables quali-

tatives pose des problèmes non résolus, car la plupart des solutions pratiques s'appuient sur des heuristiques (par exemple, un découpage *median-based*) qui ne garantissent pas l'optimalité du modèle final. Notre travail apporte une contribution fondamentale en **prouvant l'inexistence** d'un encodage numérique parfaitement optimal pour les variables catégorielles avec le critère MAE. Cette preuve représente une avancée notable, compte tenu des efforts déployés pour trouver un encodage idéal adapté à la MAE et des multiples implémentations d'encodage automatique dans des frameworks tels que *XGBoost*, *LightGBM* ou *CatBoost*.

Par ailleurs, nous décrivons un nouvel algorithme qui résout efficacement le **problème du coût unimodal 2-median**, problématique étroitement liée à la construction d'arbres de décision de type CART. Au-delà de son utilité immédiate dans le contexte de notre recherche, cet algorithme s'avère pertinent pour d'autres applications en optimisation et en analyse de données.

En définitive, cette thèse vise à **améliorer** à la fois la **théorie** et la **pratique** des arbres de décision dans le domaine du *machine learning*. D'une part, nous répondons aux questions d'interprétabilité via la proposition de l'algorithme *Linear TreeShap*, qui aborde les limites computationnelles des méthodes d'explicabilité. D'autre part, nous explorons en profondeur la gestion des variables catégorielles en proposant de nouvelles méthodes de scission et en dévoilant des résultats théoriques fondamentaux quant à l'impossibilité d'obtenir un encodage optimal sous le critère MAE. Par ces contributions, nous espérons ouvrir la voie à des **arbres de décision** plus flexibles, plus puissants et plus transparents, capables de gérer efficacement une diversité de problèmes réels, qu'il s'agisse de la classification multiclasse complexe ou de la régression robuste à grande échelle.

Notre ambition est donc de "*favoriser une plus large adoption*" des arbres de décision en tant qu'outils d'apprentissage automatique, en soulignant leurs capacités à s'adapter à la complexité croissante des données tout en maintenant une certaine lisibilité pour l'utilisateur final. Ainsi, ce travail trouve une résonance non seulement dans la recherche académique mais aussi dans le déploiement pratique des modèles de *machine learning*, afin de con-

tribuer à des prises de décision plus éclairées et plus fiables dans des secteurs variés (finance, santé, e-commerce, etc.). Les évolutions proposées ici constituent autant de réponses aux défis actuels en matière de modélisation de données structurées et de traitement de variables qualitatives, ouvrant la voie à de nouvelles perspectives pour l'interprétabilité et l'efficacité computationnelle dans le domaine de l'intelligence artificielle.

Abstract

In the dynamic landscape of machine learning, where algorithms continuously strive to outpace one another, decision tree models have consistently stood out for their efficiency and transparency, particularly in handling structured data. These models excel in scenarios where human interpretability is as crucial as predictive accuracy. Despite their enduring appeal, decision trees face persistent challenges, notably in deciphering complex tree structures and efficiently managing categorical data.

Think of a decision tree model as a flowchart that systematically guides a sequence of questions toward a final decision, such as predicting whether a person is likely to purchase a product online based on their browsing behavior. Decision trees excel in this task because they pose simple, intuitive questions like, "Did the person visit the homepage?" or "Did they click on the 'Sale' section?" However, as the model aims for higher predictive accuracy, the tree becomes deeper, branching into more questions and mirroring a more intricate decision-making process.

As these trees grow, their complexity increases, making it more challenging to interpret why the model makes specific predictions. To address this complexity, recent advancements have integrated decision trees with Shapley values and TreeShap, methodologies that assign importance scores to each decision point in the tree. These techniques provide a clearer understanding of the rationale behind a decision tree's predictions, even in the context of highly complex models. This approach is akin to unraveling the layers of the model's decision-making process. However, these methods often come with significant computational costs, which can hinder the efficiency of analyzing and interpreting tree-based

models. To overcome this challenge, we propose the "Linear TreeShap" algorithm, designed to streamline the interpretation process and enable a more efficient understanding of deep tree structures.

Another critical challenge faced by decision trees is their handling of categorical features. In their predictive quest, decision trees must split features into distinct groups, akin to separating apples from oranges. Numerical features can be split naturally into ordered groups, making the task relatively straightforward. However, categorical features, such as colors, present a unique challenge. Categories are discrete and lack inherent order, leading to a proliferation of potential splits. For example, if we categorize fruits by colors like "red," "green," and "yellow," there is no natural sequence or hierarchy among these values. As the number of categories increases, the number of possible splits grows exponentially, complicating the decision tree's ability to manage categorical data effectively.

These challenges highlight the need for innovative approaches to maintaining decision trees' strengths while addressing their limitations. By developing solutions like the "Linear TreeShap" algorithm and exploring efficient categorical feature splitting methods, we aim to enhance the interpretability and computational efficiency of decision tree models, ensuring they remain valuable tools in the evolving field of machine learning.

This thesis delves into the challenges of handling categorical data within decision tree algorithms, with a particular focus on Classification and Regression Trees (CART). Traditional approaches often rely on numerical encoding methods, transforming categorical features into numerical values. While effective in binary classification scenarios, these methods frequently fall short in multi-class problems, where the task is to predict multiple outcomes, such as different colors (e.g., red, green, yellow). We propose a comprehensive framework enabling decision trees to directly split on categorical features, closing the accuracy gap between binary and multi-class problems. This framework represents a sophisticated solution, allowing decision trees to work with categories directly without necessitating their conversion into numerical counterparts. By exploring the complexities of multi-class classifications, this framework offers deeper insights into the inherent challenges of predicting

multiple outcomes.

To further enhance the efficiency of categorical feature handling, we introduce the "BSplitZ" method, a stochastic approach tailored to simplify the splitting of large sets of categories. This method effectively manages categorical data, even in scenarios with extensive and diverse category sets, thereby streamlining the process for large-scale applications.

A critical aspect of this research addresses the binary splitting of categorical features using the Mean Absolute Error (MAE) criterion. Binary splitting is a fundamental component of tree learning algorithms, and it often serves as a bottleneck, particularly when dealing with categorical features. The discrete nature of categorical features introduces an exponential search space for potential binary splits, posing significant computational challenges. Various numerical encoding methods have been developed to mitigate these, allowing popular tree-based machine learning frameworks, such as XGBoost, LightGBM, and CatBoost, to support categorical data through automatic numerical encoding techniques. While optimal numerical encodings exist for certain splitting criteria, such as mean squared error and Gini impurity, the MAE criterion—favored for its robustness against outliers and skewed distributions—lacks a well-established optimal encoding method. The most common approach for MAE employs a median-based heuristic, which, although computationally efficient, may not yield optimal results, especially in the context of large, randomly generated datasets. Proving the non-optimality of this heuristic has been particularly challenging, as verifying optimality in extensive random datasets is practically unfeasible.

Addressing these challenges, our research provides the first conclusive proof of the non-existence of optimal numerical encoding for categorical features when using the MAE criterion. This significant finding has profound implications, especially considering the substantial efforts devoted to identifying such encodings and the development of various unsupervised numerical encoding methods.

Moreover, the research introduces a novel algorithm that efficiently resolves the unimodal cost 2-median problem. This algorithm is pertinent to the context of CART algorithms and represents a valuable contribution to solving a broader computational problem with in-

dependent significance.

Collectively, this work offers novel insights and solutions to the challenges associated with handling categorical features in decision tree-based machine learning. By addressing the optimality of numerical encoding, introducing innovative algorithms, and enhancing the theoretical and practical aspects of decision tree modeling, this research advances both the foundational understanding and the application of these models.

In summary, this thesis contributes significantly to decision tree-based machine learning by presenting new algorithms, frameworks, and a deeper understanding of key issues such as interpretability, categorical feature handling, and binary splitting. We aim to advance the theoretical underpinnings of decision tree models and facilitate their practical applications across diverse domains. By illuminating the complex interplay between decision trees and structured data, this thesis aspires to bridge the gap between sophisticated algorithms and their practical utility, benefiting both experts and newcomers to the field alike.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In machine learning, decision trees have become fundamental tools, widely used in classification and regression tasks [9, 58]. Among these, the Classification and Regression Tree (CART) algorithm, initially introduced by Breiman [9] and later refined by Rokach [58], has established itself as a cornerstone. Its success has paved the way for a diverse range of tree-based algorithms that form a critical component of modern machine learning.

Decision trees are valued for their interpretability and versatility in the current machine learning landscape. They effectively model classification and regression tasks using intuitive, rule-based structures, making them applicable in a variety of fields such as financial risk assessment and medical diagnosis [9, 62]. The appeal of decision trees lies in their simplicity and transparency—features that are increasingly important as machine learning models are applied in high-stakes domains, where understanding model decisions is crucial [22, 37].

In industry, the practical applicability of decision trees has led to their widespread adoption in real-time decision-making systems, where quick and interpretable decisions are essential. Decision trees are used extensively to automate processes across various sectors,

such as credit scoring in finance and fraud detection in e-commerce [11]. The integration of decision trees into large-scale machine learning frameworks like XGBoost, LightGBM, and CatBoost has enhanced their robustness and scalability, enabling them to handle large datasets efficiently while maintaining high accuracy [13, 52].

In academia, decision trees continue to be both a practical tool and a subject of research. Efforts are ongoing to improve their performance, scalability, and interpretability, particularly in addressing the challenges posed by large and complex datasets. Recent advances have focused on enhancing the computational efficiency of tree-based algorithms, handling high-dimensional data, and integrating decision trees with other machine learning paradigms [22, 39]. However, challenges remain, such as improving the handling of categorical features, enhancing the interpretability of ensemble methods, and refining the theoretical foundations of certain decision tree practices.

Despite the widespread adoption of decision trees in academia and industry, several challenges remain. As models increase in complexity, especially with ensemble methods like Random Forests and Gradient Boosting Machines, the interpretability of individual trees decreases, raising transparency concerns [11, 49]. Additionally, improving the computational efficiency of tree-based algorithms is crucial, particularly for environments that involve large datasets or real-time processing [13, 33, 52]. There are also theoretical gaps, such as the optimal encoding of categorical features, which need more rigorous exploration to ensure best practices [39].

These challenges highlight the importance of ongoing research in decision tree methods, especially in balancing interpretability with computational efficiency. As the machine learning field increasingly prioritizes accurate and explainable models, addressing the unresolved issues in decision tree applications becomes essential.

With this context in mind, this thesis addresses three key research challenges to advancing the state of the art in decision tree methods. These challenges include efficient computation of decision tree Shapley values, developing novel methods for splitting categorical features using convex splitting criteria, and exploring non-convex splitting criteria

such as the Mean Absolute Error (MAE).

1.2 Research Challenges

1.2.1 Research Challenge 1: Efficient computing of decision tree Shapley value

The demand for explainability in machine learning has increased significantly, driven by the needs of businesses, regulators, and society at large. Various methods have been developed to enhance the interpretability of complex tree models. Notable approaches include sampling-based techniques like Local Interpretable Model-Agnostic Explanations (LIME) [56] and game-theoretical methods such as the Shapley value [68]. The Shapley value, particularly through its implementation as TreeShap [42], has been widely adopted due to its consistency and efficiency in explaining model predictions both locally and globally.

The Shapley value provides a robust mathematical framework for quantifying the contribution of each feature to a prediction, making it a key tool for achieving transparency and fairness in decision-making processes. However, as its adoption grows, there is a pressing need for more efficient implementations. Recent advancements, such as GPUTreeShap [48] and FastTreeShap [77], have focused on accelerating TreeShap computations. GPUTreeShap utilizes the parallel processing capabilities of GPUs, while FastTreeShap improves efficiency through caching mechanisms. Despite these improvements, the complexity of these methods can pose challenges in understanding and implementation.

To address the need for both efficiency and transparency, we propose a bottom-up approach for reevaluating Shapley value computation. Our method deconstructs the computation into smaller, understandable units called decision rules, leveraging the common properties of edge-based Shapley value computation and the mathematical properties of polynomial arithmetic. A key innovation in our approach is the introduction of a jump list data structure, which facilitates the cancellation of common feature polynomials, even when features appear with different edge polynomials. This leads to the development of the "Linear

TreeShap" algorithm, which computes exact Shapley values in linear time while maintaining memory efficiency. This approach provides a more straightforward and efficient method for calculating feature contributions in decision trees.

1.2.2 Research Challenge 2: Efficient Splitting categorical features with convex splitting criteria

Another significant challenge in decision tree algorithms is the handling of categorical features. While splitting numerical features is straightforward, the complexity increases considerably with categorical attributes [9, 58]. For numerical features, the number of potential two-way splits is directly related to the number of unique values, making the process manageable. In contrast, categorical features introduce a surge in complexity, often resulting in an exponential increase in the number of possible splits relative to the number of unique categories.

Modern machine learning frameworks, such as XGBoost, LightGBM, and CatBoost, are highly effective at managing numerical features but typically require users to handle feature encoding or use built-in methods that may not generalize well, especially in multi-class classification settings [13, 33, 52]. The transition from binary to multi-class classification often reveals limitations in these encoding strategies, highlighting a gap that our research aims to address.

Our goal is to reduce the complexity of multi-class categorical feature splitting. To this end, we introduce a novel state space based on additive sufficient statistics, inspired by the mathematical properties of Zonotopes, which are geometric objects extensively studied for their useful characteristics in optimization. This approach allows us to redefine the optimization landscape, thereby reducing the computational complexity associated with categorical splits [12].

This led to the development of the "BSplitZ" method, which directly addresses the complexities of multi-class categorical feature splitting while maintaining computational efficiency. "BSplitZ" employs a stochastic approach to optimize the splitting process, providing bounds

on the expected sample size. This ensures that the method remains efficient even when applied to datasets with numerous classes.

"BSplitZ" represents a significant advancement in decision tree-based machine learning by specifically addressing the complexities associated with multi-class categorical feature splitting. By balancing precision and efficiency, it sets a new standard for handling categorical data in decision tree models. This method not only enhances accuracy but also offers a practical and accessible solution for both researchers and practitioners dealing with complex categorical datasets.

1.2.3 Research Challenge 3: Efficient Splitting categorical features with non-convex splitting criteria MAE

The Mean Absolute Error (MAE) criterion plays a significant role in decision tree modeling, particularly when dealing with data that includes outliers and skewed distributions due to its robustness and reliability. While the MAE criterion is commonly utilized across various statistical domains, a key challenge has been identifying an optimal numerical encoding method for categorical features within this framework.

Our research investigates this challenge in depth, with a focus on understanding the nuances of MAE criteria and numerical encoding. The most prevalent numerical encoding method for MAE relies on a median-based heuristic, known for its efficiency with a time complexity of $O(n \log n)$, where n is the size of the dataset. Despite its widespread use, this heuristic is potentially sub-optimal for large, randomly generated datasets, a limitation that has been difficult to prove due to the impracticality of exhaustive manual verification formally [73].

In chapter 5, we demonstrate the non-optimality of the median-based heuristic and establish a significant result: no optimal numerical encoding method exists for splitting categorical features with the MAE criterion. This finding highlights the inherent complexity of the problem and has important implications for decision tree modeling.

To address this challenge, we introduce an exact algorithm specifically designed for the

binary splitting of categorical features under the MAE criterion. This algorithm offers high accuracy and achieves computational efficiency comparable to leading heuristics, thereby providing a practical solution to the challenges associated with MAE criteria and numerical encoding. This contribution fills a critical gap in the understanding and application of decision tree modeling, facilitating more accurate and efficient handling of categorical features in real-world scenarios.

Our research extends beyond theoretical insights to provide actionable solutions to practical challenges in decision tree-based machine learning. Whether through the enhanced efficiency of categorical feature splitting with "BSplitZ," the improved interpretability of deep tree structures with "Linear TreeShap," or the identification of limitations in optimal numerical encoding for the MAE criterion, our work has the potential to make a significant impact on the field. These contributions, spanning computational efficiency, interpretability, and foundational encoding methods, are poised to benefit both researchers and practitioners, advancing the application of machine learning in diverse real-world contexts.

1.3 Contributions

This section summarizes this thesis's contributions to provide a clear understanding for readers at different levels of familiarity with the subject matter.

1.3.1 Linear Tree Shap: A Path to Efficiency and Transparency

Our primary contribution is the development of the "Linear Tree Shap" algorithm, which represents a significant advancement in model interpretability within machine learning. This algorithm efficiently computes Shapley values, a crucial concept for assessing feature contributions in decision trees.

The key innovation of the "Linear Tree Shap" algorithm is its enhanced computational efficiency. While previous methods were constrained by a complexity proportional to the square of the tree's depth (D^2), limiting their scalability, our approach reduces this complex-

ity to linear with respect to the tree's depth (D). This improvement makes Shapley value calculations substantially faster and more feasible for large-scale models.

In addition to efficiency, "Linear Tree Shap" addresses the critical need for transparency in machine learning, where the interpretability of models is as important as their accuracy. By providing a clear understanding of feature contributions, the algorithm supports informed decision-making and accountability in high-stakes applications, such as finance and healthcare [17].

The practical benefits of "Linear Tree Shap" are validated through rigorous empirical studies, demonstrating that the theoretical efficiency gains are realized in practical applications. These studies confirm that the algorithm offers a reliable tool for practitioners, balancing both efficiency and transparency in decision tree models.

1.3.2 Bridging the Gap in Binary Class and multi-class Classification for Categorical Feature Splitting: The BSplitZ Method

Handling categorical features in decision tree learning presents notable challenges, especially when moving from binary to multi-class classification. Our research addresses this gap by proposing efficient solutions to improve the handling of categorical features.

Contribution 1: A Novel Formulation for Categorical Feature Encoding

Our first contribution introduces a novel formulation that provides a clear and concise proof of the numerical encoding mechanism for categorical features in the context of two-class classification and regression. This formulation aims to simplify and clarify this important aspect of machine learning, making it more accessible and understandable.

Contribution 2: Polynomial Complexity in Multi-Class Classification

We challenge the prevalent belief that splitting categorical features leads to exponential complexity with the number of categories. Our research demonstrates that this complexity

increases polynomially with respect to the number of categorical features. This insight has important implications for decision tree modeling, facilitating more efficient and accurate multi-class classification without overwhelming computational demands.

Contribution 3: The BSplitZ Method

The core contribution of our work is the "BSplitZ" method, which specifically addresses the challenges of multi-class categorical feature splitting. BSplitZ employs a stochastic approach to optimize the splitting process in n-class classification, providing bounds on the expected sample size to ensure computational efficiency.

Practically, "BSplitZ" offers an effective and precise solution for decision tree modeling in multi-class scenarios. It addresses the complexities of categorical feature splitting, delivering a practical and scalable method suitable for datasets with a large number of classes. This development enhances approaches to multi-class classification involving categorical features, making the process more efficient and accessible to both practitioners and researchers.

1.3.3 Unveiling the Mystery of Splitting Categorical Features with Mean Absolute Error (MAE) Criteria

Mean Absolute Error (MAE) criteria are effective in decision tree modeling, particularly when dealing with data containing outliers and skewed distributions. Known for their robustness and accuracy, MAE criteria are widely used in various statistical domains. However, a persistent challenge has been the development of an optimal numerical encoding method for categorical features under MAE.

Contribution 1: Exploring Numerical Encoding for Categorical Features with MAE

Our research investigates this challenge in depth, revealing that no unsupervised numerical encoding method is optimal for MAE. This finding highlights the inherent complexity of the problem and suggests that the pursuit of an optimal encoding strategy may be inherently limited by the nature of MAE itself.

Contribution 2: Development of an "Exact" Algorithm

To address the challenges associated with the MAE criterion, we developed an exact algorithm tailored for the binary splitting of categorical features. This algorithm achieves efficiency comparable to leading heuristics, offering a practical and reliable solution to the challenges posed by MAE criteria and numerical encoding. It provides decision tree modelers with a robust tool to navigate the complexities of datasets, enhancing their ability to make informed, data-driven decisions.

Our contributions are not confined to theoretical exploration; they are designed with real-world applicability in mind. We focus on delivering concrete solutions to decision tree modelers that address the intricate challenges posed by the MAE criterion, ensuring practical utility in complex decision-making scenarios.

In summary, our work offers practical advancements to key challenges in decision tree-based machine learning. From improving the efficiency and transparency of Shapley value calculations with "Linear Tree Shap" to enhancing multi-class categorical feature splitting with "BSplitZ" and resolving the complexities of MAE criteria with our exact algorithm, our contributions have the potential to significantly influence the field. These advancements provide valuable tools for both researchers and practitioners, advancing the practical application of machine learning in diverse real-world contexts.

1.4 Thesis Outline

This thesis is structured into six chapters, each addressing key challenges and contributions in decision tree-based machine learning. Chapter 1 introduces the motivations behind the research and outlines the primary challenges, including efficient computation of decision tree Shapley values, splitting categorical features with convex criteria, and handling non-convex splitting with the MAE criterion. It also details the thesis's main contributions, such as the development of the Linear Tree Shap algorithm, the BSplitZ method for multi-class categorical splitting, and novel solutions for the MAE criterion.

Chapter 2 reviews the relevant literature, covering advances in Shapley value computation, optimal decision tree methods, and the complexities of categorical feature splitting. Chapter 3 to chapter 5 delve into the core contributions: chapter 3 focuses on the Linear Tree Shap algorithm, chapter 4 introduces the BSplitZ method for efficient multi-class feature splitting, and chapter 5 addresses the MAE criterion, providing new insights and an innovative algorithm for binary splitting. Chapter 6 concludes the thesis by summarizing the findings, highlighting the impact of the research, and suggesting future research directions. The thesis also includes a list of related publications showcasing the dissemination of the research findings.

1.4.1 Publications

The research work presented in this thesis has been published in scientific venues in the corresponding field. In the following, we provide a list of selected publications:

— Peng Yu, Albert Bifet, Jesse Read, Chao Xu. "Linear tree shap" In: 2022 Advances in Neural Information Processing Systems, pp. 25818–25828

The following papers have been submitted to journals in the field of machine learning and are under review:

— Peng Yu, Albert Bifet, Jesse Read. "Improved Binary Splitting of Categorical Features in Decision Tree Learning" In: (2023). Submitted

— Peng Yu, Chao Xu, Albert Bifet, Jesse Read. "Binary Split Categorical feature with Mean Absolute Error Criteria in CART" In: (2023). Submitted

Chapter 2

Background and Related Work

Chapter Overview for Non-Experts. This chapter aims to equip readers with both conceptual and methodological tools needed to understand the subsequent contributions of this thesis. We begin by establishing essential definitions and theoretical underpinnings for decision tree explainability (Section 2.1), then proceed to discuss how categorical data is handled in tree-based models with convex splitting criteria(Section 2.2) and introduce Mean Absolute Error (MAE) as a more challenging non-convex criterion (Section 2.3). Throughout, we provide illustrative examples and concise mathematical formulations to ensure a clear foundation for non-expert readers.

2.1 Related Work on Linear Complexity Algorithms for Tree-SHAP

Efficient computation of Shapley values [68] for decision trees is a critical focus in interpretable machine learning. Shapley values offer a principled framework for quantifying the contribution of individual features to model predictions. This methodology, while originally conceived for cooperative game theory, has proven valuable in interpreting machine learning models, especially in complex scenarios involving deep or ensemble decision trees.

Understanding feature contributions is particularly crucial in high-stakes applications such as finance and healthcare, where model transparency and trustworthiness are paramount [60].

Despite the theoretical robustness of Shapley values, their computation is often resource-intensive, requiring the evaluation of all possible feature combinations to estimate contributions accurately [72]. This complexity is exacerbated in large decision trees, where potential feature interactions grow exponentially. Consequently, a significant body of work has emerged aiming to leverage the unique structure of decision trees to develop algorithms that compute Shapley values with polynomial computational complexity. These advancements balance computational efficiency and precision, making Shapley value estimation feasible in large-scale machine learning systems [43].

2.1.1 Shapley Values and Their Role in Interpretability

Shapley values, grounded in cooperative game theory, provide a rigorous mathematical framework for quantifying feature importance in machine learning models. In decision trees, Shapley values measure the marginal contribution of each feature by comparing predictions with and without that feature [65]. This method captures both individual and interactive effects, making it a powerful tool for interpretability, especially in complex models with intricate feature interactions [14].

Their relevance extends beyond transparency to practical applications such as feature selection and model debugging [71]. For example, Shapley values enable data scientists to identify which features drive predictions, facilitating more informed decision-making during model refinement. Additionally, the consistent and fair attribution of feature importance makes them particularly suitable for sensitive applications, where accountability and fairness are essential [22].

2.1.2 Related Work on Optimal Decision Trees

Optimal decision trees have garnered attention for enhancing accuracy and interpretability, contributing to the broader context of decision tree-based models. Although not the primary focus of this thesis, their development is relevant. Bertsimas and Dunn [5] introduced optimal classification trees aimed at minimizing classification error while maintaining interpretability constraints. This approach has shown that decision trees can be optimized to achieve high predictive accuracy without sacrificing transparency, which is crucial in real-world applications.

Other works have sought to balance model complexity and accuracy. For instance, Blanquero et al. [6] proposed sparse optimal regression trees that incorporate regularization techniques to enhance generalization while maintaining interpretability. Similarly, Verwer and Zhang [75] developed optimal classification trees using binary linear programs, making the models computationally feasible for larger datasets.

Despite these advancements, optimal decision trees often face challenges, such as increased computational demands and the risk of overfitting [64]. Therefore, regularization techniques and pruning strategies are essential to maintain the balance between accuracy and interpretability. These considerations underscore the complexity of designing decision tree models that are both optimal and practical for real-world use.

2.1.3 Importance of Shapley Values in Decision Trees

Shapley values enhance the interpretability of decision trees in several ways:

— **Feature Selection**: By quantifying each feature's contribution, Shapley values support more informed feature selection, especially in high-dimensional datasets, where selecting relevant features is crucial for both performance and interpretability [46].

— **Model Explainability**: Shapley values further enhance the explainability of decision trees, particularly for complex or deep trees. They provide clear attribution of feature contributions, helping communicate model behavior to non-experts, thereby fostering trust and accountability [42].

— **Bias and Fairness Assessment**: Shapley values help assess decision tree fairness by identifying features that disproportionately influence predictions, enabling the detection and mitigation of biases—a growing concern in machine learning, especially in sensitive domains like hiring or criminal justice [29].

Thus, Shapley values significantly contribute to making decision trees transparent and reliable in practical applications.

2.1.4 Linear TreeSHAP: Achieving Linear Computational Complexity

A key advancement in the efficient computation of Shapley values is the Linear TreeSHAP algorithm [78]. Unlike traditional methods with exponential or quadratic complexity, Linear TreeSHAP exploits the hierarchical structure of decision trees to achieve linear complexity in the number of nodes. This breakthrough enables the application of Shapley values to large, complex models without prohibitive computational costs.

By preserving the accuracy of Shapley value computations while dramatically reducing computational time, Linear TreeSHAP makes Shapley-based interpretability feasible in real-world applications, where both accuracy and efficiency are critical [42]. This represents a significant leap in making Shapley values a viable tool for large-scale machine learning systems.

2.1.5 The Evolution of TreeSHAP Algorithms

Before the development of Linear TreeSHAP, several TreeSHAP variants were proposed to improve computational efficiency. Notably, GPU TreeSHAP [48] and FastTreeSHAP [77] introduced optimizations that leveraged hardware and algorithmic improvements to accelerate Shapley value computations. While these methods contributed to reducing computation time, the linear complexity achieved by Linear TreeSHAP represents a substantial advancement.

2.1.6 Sampling-Based Strategies: Balancing Speed and Precision

Sampling-based methods have been explored as an alternative solution to the computational burden of Shapley value computation. These techniques estimate Shapley values through random sampling, significantly reducing computation time [79]. However, they introduce a trade-off between speed and precision, as approximating Shapley values through sampling sacrifices some accuracy. In scenarios where real-time performance is essential, these strategies provide a practical alternative to exact computations.

2.1.7 Interpretable Decision Trees with Linear Complexity

The development of Linear TreeSHAP aligns with the broader goal of creating interpretable decision trees with efficient computational properties. By ensuring Shapley values can be computed efficiently, Linear TreeSHAP supports interpretability, even in large and complex models. This advancement emphasizes the importance of developing methods that prioritize both transparency and scalability, facilitating the deployment of interpretable machine learning models in real-world applications [60].

In summary, Linear TreeSHAP represents a transformative step in the computation of Shapley values, offering a linear complexity solution to a previously computationally intractable problem. By balancing accuracy and efficiency, Linear TreeSHAP facilitates the practical application of interpretable decision trees in modern machine learning systems.

2.2 Related Work on Splitting Categorical Features in Multi-class Classification Tasks

Efficiently splitting categorical features in decision trees, particularly for multi-class classification, presents a significant challenge in machine learning. This section explores key approaches and their evolution, analyzing traditional decision tree algorithms, the integration of statistical tests, and modern innovations that address this complex task.

2.2.1 Traditional Decision Tree Algorithms

Traditional decision tree algorithms, such as AID (Automatic Interaction Detection) [2], THAID (Chi-squared Automatic Interaction Detection) [74], CART (Classification and Regression Trees) [9], and C4.5 [54], have laid the groundwork for handling categorical features. These algorithms aim to minimize impurity measures such as the Gini index or entropy to determine optimal splits. While foundational, they were primarily designed for binary or ordinal features and often struggle with high-dimensional, multi-class categorical data.

A well-documented limitation of traditional decision trees is their tendency to overfit, particularly with categorical data containing numerous distinct values [55]. Overfitting results in overly complex trees that capture noise rather than the underlying data patterns, compromising model generalization.

Another challenge is variable selection bias, where heuristic-based measures (e.g., Gini index, entropy) tend to favor features with more categories, leading to suboptimal splits [35]. This bias is problematic in datasets with categorical variables containing many unique values, potentially masking the true predictive capabilities of the model.

Traditional decision trees also encounter difficulties with class imbalance, especially in multi-class settings. They often skew towards the majority class, diminishing the model's ability to accurately capture minority class behaviors [18]. These limitations have driven the development of more advanced techniques for categorical feature splitting.

2.2.2 Incorporating Statistical Tests in Splitting

To address these challenges, methods incorporating statistical tests have emerged as a means to reduce biases in the splitting process. Algorithms such as FACT (Fast Algorithm for Classification Trees) [41] and QUEST (Quick, Unbiased, Efficient Statistical Tree) [38] leverage statistical tests, including the t-test and ANOVA F-test, to inform splitting decisions. These methods aim to provide more statistically grounded and unbiased feature splits.

However, despite their advantages, FACT and QUEST rely on assumptions about the underlying data distribution, often assuming normality, which is rarely met in real-world

datasets. While statistical tests reduce selection bias, they do not fully address the challenges of handling high-dimensional categorical data. Moreover, the computational cost of applying these tests at each split can be prohibitive for large datasets [39].

2.2.3 Optimal Categorical Feature Splitting

The pursuit of optimal methods for splitting categorical features has led to various innovative approaches, particularly for multi-class classification tasks. One approach involves using clustering techniques, such as k-means clustering [32], to group similar categories, reducing the complexity of the split.

While clustering-based methods offer scalability, they often sacrifice optimality. The effectiveness of k-means clustering, for example, heavily depends on selecting an appropriate number of clusters (k) and struggles with non-spherical category distributions [21]. These limitations are more pronounced in categorical data, where natural clusters might not exist, leading to splits that fail to capture meaningful distinctions.

Encoding techniques, such as one-hot encoding [26] and target encoding [47], have also been explored to represent categorical variables numerically. Target encoding, using the mean target value of each category, can be useful for binary classification but faces significant challenges in multi-class tasks. The risk of introducing noise and overfitting increases in multi-class settings due to the complexity of category-class relationships.

2.2.4 Unified Framework for Multi-class Classification

A unified approach to categorical feature splitting in multi-class classification has garnered interest. Breiman's work [8] highlighted the importance of convexity in splitting criteria for multi-class classification tasks, providing a theoretical foundation. However, practical implementations that seamlessly integrate multi-class classification with categorical feature splitting remain limited.

The complexity lies in effectively capturing relationships between categories across multiple classes. Existing methods often rely on domain-specific heuristics or oversimplified

models that fail to generalize effectively. Without a robust, generalizable framework, splitting categorical features in multi-class tasks remains challenging [3].

2.2.5 The BSplitZ Method: Leveraging Convex Zonotopes for Categorical Feature Splitting

The BSplitZ method, a key contribution of this thesis, introduces an innovative approach to categorical feature splitting in multi-class classification by leveraging the geometric properties of convex Zonotopes. Zonotopes, a subclass of polytopes characterized by symmetry and centrality, have been utilized in various fields such as computational geometry and optimization [34].

BSplitZ builds upon these properties to address the challenges of high-dimensional, multi-class categorical feature splitting. Unlike traditional methods, BSplitZ efficiently navigates the complex solution space by utilizing the symmetry inherent in Zonotopes, reducing computational complexity without sacrificing accuracy. This novel approach offers significant advantages in handling large, diverse datasets, where other methods often fall short.

By adapting the geometric properties of Zonotopes, BSplitZ ensures a more structured and principled exploration of categorical feature splits, making it a scalable and efficient solution for multi-class classification tasks. This method represents a substantial advancement in the field, demonstrating that leveraging geometric insights can lead to more effective categorical feature splitting.

2.2.6 Vertex Enumeration of Zonotopes and its Role in BSplitZ

Recent advances in vertex enumeration algorithms have provided a computational foundation for applying Zonotopes in categorical feature splitting, and BSplitZ harnesses these developments. Avis's reverse search algorithm [4] offers an efficient method for enumerating the vertices of a Zonotope, which is crucial for the BSplitZ method's implementation. This algorithm has been refined and applied in various domains, including optimization and

computational geometry [10].

Building on this foundation, BSplitZ incorporates vertex enumeration techniques to effectively determine optimal splits in high-dimensional, multi-class datasets. Stinson et al. [70] introduced probabilistic methods that further reduce the computational overhead of vertex enumeration, enabling BSplitZ to operate efficiently even with large datasets.

2.2.7 Summary

In conclusion, categorical feature splitting in multi-class classification remains an active research area. While traditional decision tree methods provide a foundational understanding, their limitations have necessitated more advanced techniques. Statistical tests, clustering, and encoding strategies have offered partial solutions, but optimality and scalability are still challenging. The introduction of geometric approaches, particularly those utilizing convex Zonotopes and vertex enumeration algorithms, represents a promising direction, offering a more principled and efficient way to handle the inherent complexity of multi-class categorical feature splitting.

2.3 Related Work on Splitting Categorical Features with the MAE Task

Binary splitting in decision tree construction is crucial for determining the structure and performance of tree-based models. When applied to categorical features, this process introduces unique challenges due to the discrete nature of the data. The Mean Absolute Error (MAE) criterion, while widely recognized for its robustness in regression tasks, presents unique challenges when applied to splitting categorical features due to its non-convex nature. Unlike most traditional approaches that rely on convex splitting criteria, the non-convexity of MAE complicates the optimization process, making it more difficult to identify optimal splits for categorical features. This adds an additional layer of complexity to

the task, requiring more sophisticated techniques to achieve effective splitting in decision tree models. . This section reviews the extensive literature on this topic, encompassing the challenges, methodologies, and ongoing advancements in the field.

2.3.1 Binary Splitting Challenges

Binary splitting is foundational in decision tree algorithms, where features are recursively split to construct a tree structure that captures the data distribution. For numerical features, algorithms such as CART [9] employ impurity measures like the Gini index or entropy to identify optimal split points. However, this complexity increases significantly when handling categorical features, particularly those with high cardinality.

A categorical feature with k unique categories has $2^{(k-1)} - 1$ potential binary splits, making exhaustive search computationally infeasible for large k. Loh and Shih [40] highlighted that this exponential increase in potential splits introduces significant computational challenges. Traditional algorithms, like C4.5 [54] handles categorical features by treating each category as a potential split point, effectively splitting the data into multiple branches—one branch for each category present in the feature. This means that for a categorical feature with k unique values, the split results in k branches, each corresponding to a specific category. While this approach allows C4.5 to handle categorical features directly, it can lead to challenges when the feature has high cardinality, resulting in a large number of branches. This can increase the risk of overfitting, as the model might create overly complex trees that fail to generalize well on unseen data [55].

Integrating the MAE criterion provides a more robust alternative by minimizing the average absolute error between predicted and actual values, making it less sensitive to outliers. However, effectively incorporating MAE into categorical feature splitting requires careful consideration of how categories are grouped and encoded to ensure accurate splits.

Example of MAE with Tree Splitting. To illustrate how Mean Absolute Error (MAE) influences a split, consider a small dataset where a categorical feature `City` has three cate-

gories: {A, B, C}, and the target is a numerical value (e.g., housing price). Suppose category A has target values {100, 110}, B has {90}, and C has {130, 135}. A binary split might group {A, B} on one branch and {C} on the other. Under MAE, each branch seeks to minimize the sum of absolute deviations from its mean prediction. For {A, B}, the optimal prediction is the median(100, 110, 90)$= 100$; for {C}, the median(130, 135)$= 132.5$. Hence the MAE for this split is $\sum |y - 100|$ over {A,B} plus $\sum |y - 132.5|$ over {C}. A different grouping, say {A} vs. {B, C}, yields another total MAE, and the best split is the one with the lowest overall MAE. This example highlights the added complexity when categories must be grouped optimally under MAE, as neither group membership nor median-based predictions are straightforward for large or diverse categories.

2.3.2 Encoding Categorical Data

A variety of encoding techniques have been developed to transform categorical features into numerical representations, facilitating their integration into decision tree algorithms. Below is a summary of some popular methods and their relevance to MAE-based splitting:

— **One-Hot Encoding**: Transforms each category into a binary vector [26]. While effective for preserving categorical nature, one-hot encoding leads to high-dimensional feature spaces, especially for features with many unique categories, which can be computationally intensive for decision tree algorithms.

— **Binary Encoding**: Offers a more compact representation by converting categories into binary codes [47]. This approach reduces dimensionality but may introduce difficulties in interpreting splits, as the binary representation doesn't capture inherent relationships between categories.

— **Ordinal Encoding**: Maps categories to integer values based on an arbitrary order [66]. Although computationally efficient, it imposes artificial ordering, potentially biasing the model, especially for categories without a natural sequence.

— **Frequency-Based Encoding**: Utilizes the frequency of categories in the dataset [24]. It aims to incorporate statistical properties but doesn't always align well with the

MAE criterion, as category frequencies may not correlate with the target variable's distribution.

— **Target Encoding**: Replaces categories with the mean target value [47]. Effective for capturing category-target relationships in binary classification, but prone to overfitting in multi-class tasks, especially when categories have few observations.

— **Hybrid Encoding Approaches**: Combine strengths of multiple methods, such as integrating one-hot encoding with feature hashing to manage dimensionality [69]. These methods have shown potential in improving decision trees' performance in MAE-based splitting but require careful calibration to prevent introducing noise.

Integrating these encoding techniques with the MAE criterion remains an active research area, as the choice of encoding directly impacts the decision tree's ability to accurately split categorical features.

Illustrative Example. To clarify how these encoding methods transform categorical features, consider a toy dataset with a single categorical feature `Color` {"Red", "Green", "Blue"} and a regression target. Table 2.1 summarizes how each encoding (One-Hot, Binary, Ordinal, etc.) converts the categories into numerical form. For instance, One-Hot generates three binary columns (`IsRed`, `IsGreen`, `IsBlue`), while Ordinal assigns integer labels (e.g., Red$\rightarrow 1$, Green$\rightarrow 2$, Blue$\rightarrow 3$) that may or may not reflect real semantic ordering. Such visualization helps to see how these encodings impact subsequent splits, especially under criteria like MAE.

Table 2.1 – Illustration of various encoding methods for a categorical feature `Color`.

`Color`	**One-Hot Encoding**	**Binary Encoding**	**Ordinal Encoding**	**Target Encoding**
Red	(1, 0, 0)	00	1	μ_{Red}
Green	(0, 1, 0)	01	2	μ_{Green}
Blue	(0, 0, 1)	10	3	μ_{Blue}

2.3.3 Comparison with Other Splitting Criteria

While MAE is valued for its robustness and interpretability, other Splitting Criteria such as Mean Squared Error (MSE) and Huber loss are also commonly used. MSE emphasizes high-magnitude errors [28], leading to splits that may be more influenced by outliers. In contrast, Huber loss balances the sensitivity between MAE and MSE, offering robustness while maintaining differentiability [30]. However, integrating these alternative loss functions into splitting tasks for categorical features remains less explored compared to MAE, particularly in high-cardinality scenarios.

2.3.4 Median-Based Heuristic Encoding

Median-based heuristic encoding has been proposed as an approach to integrate the MAE criterion into decision trees [73]. By assigning numerical values based on the median relationship with the target variable, it aims to align splits with MAE's focus on minimizing absolute errors.

While this heuristic is computationally efficient, it does not always capture the true distribution of categorical features, potentially leading to suboptimal splits. Recent work, such as [59], introduced weighted median encoding to address this limitation, enhancing alignment with the MAE criterion by considering category importance relative to the target variable.

2.3.5 Ongoing Research and Advancements

The pursuit of more effective methods for MAE-based splitting continues, with recent research focusing on several key areas:

— **Unsupervised Numerical Encoding**: Techniques such as those introduced by [45] generate numerical representations without labeled data, leveraging feature distributions to create encodings suitable for MAE-based splitting. These approaches show promise but require further validation across diverse datasets.

— **Hybrid Encoding Methods**: By combining the strengths of different encoding strate-

gies, hybrid methods aim to balance dimensionality reduction, interpretability, and alignment with the MAE criterion [69]. This approach has shown potential in enhancing decision trees' ability to handle high-cardinality features.

— **Algorithmic Developments**: Sampling-based strategies, as explored by [61], offer a way to approximate optimal split points, providing a balance between computational efficiency and split quality. Dynamic programming techniques, like those proposed by [16], also contribute to optimizing the split selection process under the MAE criterion.

— **Deep Learning Approaches**: Techniques such as entity embeddings [23] have emerged as a novel way to encode categorical features, potentially capturing complex relationships missed by traditional encoding methods. These neural network-based encodings present opportunities for enhancing MAE-based decision tree models.

2.3.6 Summary

In summary, splitting categorical features using the MAE criterion presents a complex challenge. Despite significant advancements in encoding techniques and optimization strategies, no one-size-fits-all solution has emerged. The ongoing exploration of hybrid encoding methods, unsupervised approaches, and integration with deep learning demonstrates a commitment to developing robust and effective solutions. This thesis builds on this foundation, presenting new insights into encoding strategies and their implications for decision tree construction using the MAE criterion.

Chapter 3

Linear TreeShap

3.1 Introduction

Machine learning in the industry has played more and more critical roles. The need for explainability has increased dramatically for both business and fairness purposes. As one of the most popular machine learning models, the tree-based model attracted much attention. Several methods were developed to improve the interpretability of complex tree models, such as sampling-based local explanation model LIME[56], game-theoretical based Shapley value[68], etc. Shapley value gained particular interest due to both local and globally consistent and efficient implementation: TreeShap[42]. With the broad adoption of Shapley value, the industry has been seeking a much more efficient implementation. Various methods like GPUTreeShap[48] and FastTreeShap[77] were proposed to speed up TreeShap. GPUTreeShap primarily focuses on utilizing GPU to perform efficient parallelization. And FastTreeShap improves the efficiency of TreeShap by utilizing caching. All of them are empirical approaches lacking a mathematical foundation, thus making them hard to understand.

We solve the exact Shapley value computing problem based on polynomial arithmetic. By utilizing the properties of polynomials, our proposed algorithm, Linear TreeShap, can

compute the exact Shapley value in linear time. And there is no compromise in memory utilization.

3.1.1 Contrast with previous result

We compare the running time of our algorithm with previous results for a single tree in Table 3.1, since all current algorithms for the ensemble of trees apply the same algorithm to each tree individually.

Let S be the number of samples to be explained, N the number of features, L the number of leaves in the tree, and D is the maximum depth of the tree. We assume every feature is used in the tree for simplicity, and therefore $N = O(L)$. Also, $D \leq L$.

Algorithm	Time Complexity	Space Complexity
Original TreeSHAP [42]	$O(SLD^2)$	$O(D^2 + N)$
Fast TreeSHAP v1 [77]	$O(SLD^2)$	$O(D^2 + N)$
Fast TreeSHAP v2 [77]	$O(L2^D D + SLD)$	$O(L2^D)$
Linear TreeSHAP	$O(SLD)$	$O(D^2 + N)$

Table 3.1 – Comparison of both computational and space complexity

Applicability to Deep Learning Models. While our polynomial-based approach is highly effective for decision trees, extending it directly to deep learning models is not straightforward. Deep networks can, in principle, be represented as collections of piecewise linear rules, but they typically exhibit a combinatorial explosion in the number of these rules. This complexity makes it challenging to reuse overlapping computations, which is the foundation of Linear TreeShap. Nevertheless, approximate solutions—such as distilling neural networks into compact tree ensembles—may benefit from our approach by then applying Linear TreeShap on the distilled models.

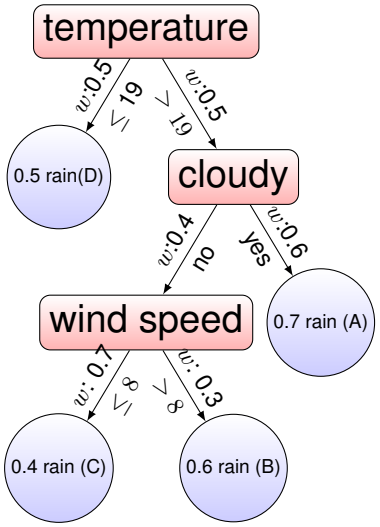

Figure 3.1 – An example decision tree T_f shows chances of rain

3.2 Methodology

3.2.1 Notation & Background

Elementary symmetry polynomials are widely used in our paper. $\mathbb{R}[x]$ denotes the set of polynomials with coefficient in $\mathbb{R}$. $\mathbb{R}[x]_d$ are polynomials of degree no larger than d. We use $\odot$ for polynomial multiplication. For two polynomials a and b, $\lfloor \frac{a}{b} \rfloor$ is the quotient of the polynomial division a/b.

For two vectors $x, y \in \mathbb{R}^d$, $\langle x, y \rangle$ is the inner product of x and y. We abuse the notation, so when a polynomial appears in the inner product, we take it to mean the coefficient vector of the polynomial. Namely, if p, q are both polynomials of the same degree, then $\langle p, q \rangle$ is the inner product of their coefficient vectors. We use $\cdot$ for matrix multiplication.

We refer to $x \in X \subset \mathbb{R}^m$ as an instance and $f : X \to \mathbb{R}$ as the fitted tree model in a supervised learning task. Here, m denotes the number of all features, M is the set of all features, and $|.|$ is the cardinality operation, namely $|M| = m$. We denote $x[i]$ as the value of feature i of instance x.

We have to start with some common terminologies because our algorithm is closely involved with trees. A rooted tree $T = (V, E)$ is a directed tree where each edge is oriented

away from the root $r \in V$. For each node v, P_v is the root to v path, i.e. the set of edges from the root to v. $L(v)$ is the set of leaves reachable from v. $L(T) = L(r)$ is the set of leaves of T. T is a full binary tree if every non-leaf node has two children. If an edge e goes from u to v, then u and v are the tail and head of e, respectively. We write $h(e)$ for the head of e.

A tree is weighted if there is an edge weight w_e for each edge e. It is a labeled tree if each edge has a label ℓ_e. For a labeled tree T, let E_i be all the edges of the tree with label i. Similarly, define $P_{i,v} = P_v \cap E_i$, the set of edges in the root to v path with label i. The last edge of any subset of a path is the edge furthest away from the root.

For our purpose, a decision tree is a weighted labeled rooted full binary tree. There is a corresponding decision tree for the fitted tree model T_f. The internal nodes of the tree are called the decision nodes, and the leaves are called the end nodes. Every decision node has a label of feature i, and every end node contains a prediction v. The label of each edge is the feature of the head node of the edge. We will call the label on the edge of the feature. Every edge e contains weight w_e, the conditional probability based on associated splitting criteria during training.

When predicting a given instance, x, decision tree model f sends the instance to one of its leaves according to splitting criteria. We draw an example decision tree in Fig 3.1. Each leaf node is labeled with an id and prediction value. Every decision node is labeled with the feature. We also associate each edge with conditional probability w and splitting criteria of features in the parenting node.

To represent the marginal effect, we use $f_S(x) : X \to \mathbb{R}, S \subseteq M$ to denote the prediction of instance x of the fitted tree model using only the features in *active set* S, and treat the rest of features of instance x as missing. Using this representation, the default prediction $f(x)$ is a shorthand for $f_M(x)$.

The *Shapley value* of a decision tree model f is the function $\varphi(f, i) : X \to \mathbb{R}$,

$$\varphi(f,i)(x) = \frac{1}{|M|} \sum_{S \subseteq M \setminus \{i\}} \frac{1}{\binom{|M|-1}{|S|}} f_{S\cup\{i\}}(x) - f_S(x). \tag{3.1}$$

The Shapley value $\varphi(f, i)(x)$ quantifies the marginal contribution of feature i in the tree model f when predicting instance x. The problem of computing the Shapley values is the following algorithmic problem.

The Tree Shapley Value Problem

Input: A decision tree T_f for function $f : X \to \mathbb{R}$ over m features, and $x \in X$.

Output: The vector $(\varphi(f, 1)(x), \ldots, \varphi(f, m)(x))$.

Meanwhile, decision nodes cannot split instances with missing feature values. A common convention is to use conditional expectation. When a decision node encounters a missing value, it redirects the instance to both children and returns the weighted sum of both children's predictions. The weights differ between decision nodes and are empirical instance proportions during training: w_l, w_r. Here $w_l + w_r = 1$ and $0 < w_l < 1$. A similar approach is also used in both Treeshap[42], and C4.5 [63] to deal with missing values. Any instance would result in a single leaf with no missing feature. In contrast, an instance might reach multiple leaves with a missing feature.

Here, we use an example instance $x =$**(temperature: 20, cloudy: no, wind speed: 6)** with tree f in Fig 3.1 to show the full process of Shapley value computing. Following the decision nodes of T_f, the prediction $f(x)$ is leaf C: 0.4 chance of raining. Now we compute the importance/Shapley value of feature **(temperature: 18)** among x for getting a prediction of 0.4 chance of rain.

The importance/Shapley value of feature **(temperature: 18)** is

$$\varphi(f, \textbf{temperature})(x) = \frac{1}{3}\Big(\frac{1}{\binom{2}{2}}\big(f_{\{\textbf{temperature, cloudy, wind speed}\}}(x) - f_{\{\textbf{cloudy, wind speed}\}}(x)\big)$$
$$+ \frac{1}{\binom{2}{1}}\big(f_{\{\textbf{temperature, wind speed}\}}(x) - f_{\{\textbf{wind speed}\}}(x) + f_{\{\textbf{temperature, cloudy}\}}(x) - f_{\{\textbf{cloudy}\}}(x)\big)$$
$$+ \frac{1}{\binom{2}{0}}\big(f_{\{\textbf{temperature}\}}(x) - f_{\emptyset}(x)\big)\Big)$$

To elaborate more, a term $f_{\{\textbf{cloudy, wind speed}\}}(x)$ with $x =$**(temperature: 20, cloudy: no, wind speed: 6)** is equivalent to $f($**cloudy: no, wind speed: 6**$)$. When traversing the first decision node, temperature, the current feature's value is considered unspecified. We sum over children leaves with empirical weights and get $0.5 \cdot D + 0.5 \cdot C$ as the prediction.

On the other hand, decision tree f can be linearized into decision rules [53]. A decision rule can be seen as a decision tree with only a single path. A decision rule $R^v : X \to \mathbb{R}$ for a leaf v can be constructed by starting from the root node, following all the conditions along the path to the leaf v. We use $F(R)$ to represent the set of all features specified in decision rule R, namely, $F(R) = \{i | P_{i,v} \neq \emptyset\}$.

The linearization of the decision tree f to decision rules is the relation $f(x) = \sum_{v \in L(T_f)} R^v(x)$. Example tree in Fig 3.1 can be linearized into four rules:

1. R^A: **if** (temperature $>$ 19) and (is cloudy) **then** predict 0.7 chance of rain **else** predict 0 chance of rain
2. R^B: **if** (temperature $>$ 19) and (is not cloudy) and (wind speed $>$ 8) **then** predict 0.6 chance of rain **else** predict 0 chance of rain
3. R^C: **if** (temperature $>$ 19) and (is not cloudy) and (wind speed $\leq$ 8) **then** predict 0.4 chance of rain **else** predict 0 chance of rain
4. R^D: **if** (temperature $\leq$ 19) **then** predict 0.5 chance of rain **else** predict 0 chance of rain

For a decision rule R, we also introduce *prediction with active set* S, $R_S : X \to \mathbb{R}$. When features are missing, leaf value further weighted by their conditional probability, is provided as the prediction. Here, we introduce the definition recursively. First, we define the prediction of rule R associated with leaf value $\mathcal{V}$ with empty input:

$$R^v_\emptyset = R^v_\emptyset(x) = \mathcal{V} \prod_{e \in P_v} w_e \tag{3.2}$$

Where w_e is the conditional probability/proportion of instances, when splitting by decision node at the source of edge e, the proportion of instances belong to the current edge. $\mathcal{V}$ is the prediction of the leaf node v that defines that decision rule.

$$\forall a, b \in \mathbb{R}, \varphi(af + bg, i) = a\varphi(f, i) + b\varphi(g, i)$$

We say $x \in \pi_i(R)$ if $x[i]$ is satisfied by every splitting criteria concerning feature i in decision rule R. For a given instance x and leaf v, we use a new variable $q_{i,v}(x)$ to denote the marginal prediction of R^v when adding feature i to active set S.

$$q_{i,v}(x) := \begin{cases} \prod_{e \in P_{i,v}} \frac{1}{w_e} & x \in \pi_i(R^v) \\ 0 & x \notin \pi_i(R^v) \end{cases} \tag{3.3}$$

The empty product equals 1, hence if $P_{i,v} = \emptyset$, $q_{i,v}(x) = 1$. We omit the super/subscript v if there is no ambiguity on the leaf node. So, with $i \notin S$, we can write:

$$R_{\{i\} \cup S}(x) = q_i(x) R_S(x) \tag{3.4}$$

Since $\emptyset$ is a subset of any set S, we can get $R_S(x)$ via products of weights:

$$R_S(x) = R_\emptyset \prod_{j \in S} q_j(x) \tag{3.5}$$

With R_S, f_S can also be linearized into the sum of rule predictions:

$$f_S(x) = \sum_{v \in L(T_f)} R_S^v(x). \tag{3.6}$$

3.2.2 Some special functions and their properties

Definition 3.2.1. *Define the reciprocal binomial polynomial to be* $B_d(x) = \sum_{i=0}^{d} \binom{d}{i}^{-1} x^i$.

Definition 3.2.2. *The function* $\psi_d : \mathbb{R}[x]_d \to \mathbb{R}$ *is defined as*

$$\psi_d(A) := \frac{\langle A, B_d \rangle}{d+1}. \tag{3.7}$$

We write $\psi(p) = \psi_d(p)$ *where* d *is the degree of* p.

The function ψ_d has two nice properties: additive for same degree polynomial and "scale" invariant when multiplying binomial coefficient.

Proposition 3.2.1. *Let $p, q \in \mathbb{R}[x]_d$, and $k \in \mathbb{N}$.*

— *Additivity:* $\psi_d(p) + \psi_d(q) = \psi_d(p + q)$.

— *Scale Invariant:* $\psi(p \odot (1 + y)^k) = \psi(p)$.

3.2.3 Summary polynomials and their relation to Shapley value

Consider we have a function f represented by a decision tree T_f. We want to explain a particular sample x; in the later sections, we abuse the notation and let g to mean $g(x)$ whenever $g : X \to \mathbb{R}$, e.g. $q_{i,v} = q_{i,v}(x)$. Our polynomials always have the formal variable y to avoid confusing readers.

Since tree prediction can be linearized into decision rules, and the Shapley value also has Linearity property, we decompose the Shapley value of a tree as the sum of the Shapley value of decision rules.

$$\varphi(f, i) = \sum_{v \in L(T_f)} \varphi(R^v, i) \tag{3.8}$$

Now, for each decision rule, we define a summary polynomial.

Definition 3.2.3. *For a decision tree T_f and an instance x. For a decision rule associated with leaf v in T_f, the* summary polynomial G_v *is defined as*

$$G_v(y) = R^v_\emptyset \prod_{j \in F(R^v)} (q_{j,v} + y) \tag{3.9}$$

Next, we study the relationship between the summary polynomial and the Shapley value of the corresponding decision rule.

Lemma 3.2.2. *Let v be a leaf in T_f, then*

$$\varphi(R^v, i) = (q_{i,v} - 1)\psi\left(\frac{G_v}{q_{i,v} + y}\right).$$

Proof. Since everything involved in the proof is related to the leaf v, we drop v from the super/subscripts for simplicity. First, we simplify the Shapley value of rule R into:

$$\varphi(R, i) = \frac{1}{m}\sum_{S\subset M\setminus\{i\}} \frac{1}{\binom{m-1}{|S|}} R_{S\cup\{i\}} - R_S = \frac{R_\emptyset(q_i - 1)}{m}\sum_{S\subset M\setminus\{i\}} \frac{1}{\binom{m-1}{|S|}}\prod_{j\in S} q_j \tag{3.10}$$

When feature i does not appear in R, $q_i - 1$ returns 0, and the Shapley value on feature i from rule R is 0. Let $|F(R)| = d$, the number of features in R. The Shapley value of R further reduces to:

$$\varphi(R, i) = \frac{R_\emptyset(q_i - 1)}{d}\sum_{k=0}^{d-1} \frac{1}{\binom{d-1}{k}} \sum_{S\subset F(R)\setminus\{i\}}^{|S|=k}\prod_{j\in S} q_j \tag{3.11}$$

We observe that $R_\emptyset \sum_{S\subset F(R)\setminus\{i\}}^{|S|=k}\prod_{j\in S} q_j$ is precisely the coefficient of y^k in $\frac{G}{q_i+y}$.

We obtain the weighted sum of all subsets' decision rule prediction using the inner product:

$$R_\emptyset \sum_{S\subset F(R)\setminus\{i\}} 1/\binom{d-1}{|S|} \sum_{S\subset F(R)\setminus\{i\}}^{|S|=k}\prod_{j\in S} q_j = \langle\frac{G}{q_i + y}, B_{d-1}\rangle \tag{3.12}$$

Shapley value for R has a compact form as shown in Eq.3.13.

$$\begin{aligned}\varphi(R, i) &= \frac{R_\emptyset(q_i - 1)}{d}\sum_{S\subset F(R)\setminus\{i\}} \frac{1}{\binom{d-1}{|S|}}\prod_{j\in S} q_j \\ &= \frac{(q_i - 1)}{d}\langle\frac{G}{q_i + y}, B_{d-1}\rangle \\ &= (q_i - 1)\psi\left(\frac{G}{q_i + y}\right)\end{aligned} \tag{3.13}$$

□

3.2.4 Computations

Even though we have simplified the Shapley value of a decision rule in a compact form using polynomials, it is still not friendly in computational complexity. In particular, the values $q_{i,v}$ are flat aggregated statistics and do not necessarily share terms between different rules. This makes it difficult to share intermediate results across different rules. We develop an edge-based polynomial representation to benefit from the fact that decision rules overlap.

For every edge e with feature i, we use $e^{\uparrow}$ to denote its closest ancestor edge that shares the same feature. In cases such edge does not exist, $e^{\uparrow} = \perp$. We also use $x \in \pi_u$ to represent the $x[i]$ is satisfied by all splitting criteria on feature i associated with all edges in $P_{i,u}$.

$$p_e := \begin{cases} \prod_{e' \in P_{i,h(e)}} \frac{1}{w_{e'}} & x \in \pi_{h(e)} \\ 0 & x \notin \pi_{h(e)} \end{cases} \tag{3.14}$$

We also define additionally that $p_\perp = 1$.

If e is the last edge in $P_{i,v}$ then $p_e = q_{i,v}$. This is the key to completely avoiding $q_{i,v}$ and instead switching to p_e. Our analysis will ensure that any p_e that does not correspond to $q_{i,v}$ for any v and i gets canceled out.

We show a relation between the Shapley value of a decision rule and the newly defined p_e's.

Consider an operation $\oplus_{d_1,d_2} : \mathbb{R}[x]_{d_1} \times \mathbb{R}[x]_{d_2} \to \mathbb{R}[x]_{\max(d_1,d_2)}$. The subscript is omitted when d_1, d_2 is implicit.

$$G^1 \oplus G^2 := G^1 + G^2 \odot (1+y)^{d_1 - d_2}, \tag{3.15}$$

We extend the summary polynomial to all nodes in the tree. Let $G_u = \bigoplus_{v \in L(u)} G^v$. Denote d_e as the degree of G_u, where $h(e) = u$.

Proposition 3.2.3. *Let v be a leaf in T_f, and d_v be the degree of G_v then*

$$\varphi(R^v, i) = \sum_{e \in P_{i,v}} (p_e - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \right\rfloor\right) - (p_{e^\uparrow} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_{e^\uparrow} - d_v}}{y + p_{e^\uparrow}} \right\rfloor\right).$$

Proof. Let e^* be the last edge of $P_{i,v}$. We note a few facts. $p_{e^*} = q_{i,v}$, $y + q_{i,v}$ divides G_v, and the sum is a telescoping sum. Put them together.

$$\begin{aligned}
&\sum_{e \in P_{i,v}} (p_e - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \right\rfloor\right) - (p_{e^\uparrow} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_{e^\uparrow} - d_v}}{y + p_{e^\uparrow}} \right\rfloor\right) \\
=&(p_{e^*} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_{e^*} - d_v}}{y + p_{e^*}} \right\rfloor\right) \\
=&(q_{i,v} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_{e^*} - d_v}}{y + q_{i,v}} \right\rfloor\right) \\
=&(q_{i,v} - 1)\psi\left(\frac{G_v}{y + q_{i,v}} \odot (y+1)^{d_{e^*} - d_v}\right) \\
=&(q_{i,v} - 1)\psi\left(\frac{G_v}{y + q_{i,v}}\right) \\
=&\varphi(R^v, i)
\end{aligned}$$

□

The following theorem establishes the relation between Shapley values, the summary polynomials at each node, and p_e for each edge e.

Theorem 3.2.4 (Main)**.**

$$\varphi(f, i) = \sum_{e \in E_i} (p_e - 1)\psi\left(\left\lfloor \frac{G_{h(e)}}{y + p_e} \right\rfloor\right) - (p_{e^\uparrow} - 1)\psi\left(\left\lfloor \frac{G_{h(e)} \odot (y+1)^{d_{e^\uparrow} - d_e}}{y + p_{e^\uparrow}} \right\rfloor\right)$$

Proof. Based on linearity of Shapley Value, $\varphi(f, i) = \sum_{v \in L(T_f)} \varphi(R^v, i)$.

For each rule R^v, we can scale their summary polynomial G_v to the degree of $G_{h(e)}$. Based on Proposition 3.2.3,

$$\varphi(f,i) = \sum_{v \in L(T_f)} \sum_{e \in P_{i,v}} (p_e - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \right\rfloor\right) - (p_{e\uparrow} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{(d_e - d_v)+(d_{e\uparrow} - d_e)}}{y + p_{e\uparrow}} \right\rfloor\right)$$

Observe that for any (e, v) pair, we have $e \in E_i$ and $v \in L(h(e))$ if and only if $v \in L(T_f)$ and $e \in P_{i,v}$. Hence, can order the summation by summing through the edges.

$$\varphi(f,i) = \sum_{e \in E_i} \sum_{v \in L(h(e))} (p_e - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \right\rfloor\right) - (p_{e\uparrow} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{(d_e - d_v)+(d_{e\uparrow} - d_e)}}{y + p_{e\uparrow}} \right\rfloor\right)$$

Observe that at each edge e, all summary polynomial G is scaled to the same degree d_e. According to Proposition 3.2.1, we can add the summary polynomials before evaluating using $\psi(.)$. Now, focus on the first part of the sum.

$$\begin{aligned}
\sum_{e \in E_i} \sum_{v \in L(h(e))} ((p_e - 1)\psi(\lfloor \frac{G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \rfloor) &= \sum_{e \in E_i} (p_e - 1)\psi\left(\sum_{v \in L(h(e))} \left\lfloor \frac{G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \right\rfloor\right) \\
&= \sum_{e \in E_i} (p_e - 1)\psi(\lfloor \frac{\sum_{v \in L(h(e))} G_v \odot (y+1)^{d_e - d_v}}{y + p_e} \rfloor) \\
&= \sum_{e \in E_i} (p_e - 1)\psi(\lfloor \frac{\bigoplus_{v \in L(h(e))} G_v}{y + p_e} \rfloor) \\
&= \sum_{e \in E_i} (p_e - 1)\psi\left(\left\lfloor \frac{G_{h(e)}}{y + p_e} \right\rfloor\right)
\end{aligned}$$

Using the same proof, we can also obtain

$$\sum_{e \in E_i} \sum_{v \in L(h(e))} (p_{e\uparrow} - 1)\psi\left(\left\lfloor \frac{G_v \odot (y+1)^{(d_e - d_v)+(d_{e\uparrow} - d_e)}}{y + p_{e\uparrow}} \right\rfloor\right) = \sum_{e \in E_i} (p_{e\uparrow} - 1)\psi\left(\left\lfloor \frac{G_{h(e)} \odot (y+1)^{d_{e\uparrow} - d_e}}{y + p_{e\uparrow}} \right\rfloor\right)$$

□

Algorithm 1 Obtain the summary polynomial for each node.

function COMPUTESUMMARYPOLYNOMIALS(x, v, C)
  if node v is leaf **then**
    $G[v] \leftarrow C \cdot R^v_\emptyset$
  else
    if v is not the root **then**
      $e \leftarrow$ edge with v as head
      $C \leftarrow C \odot (y + p_e(x))$
      if $e^\uparrow \neq \perp$ **then**
        $C \leftarrow \frac{C}{y + p_{e^\uparrow}(x)}$
      end if
    end if
    $l, r \leftarrow$ children of v
    COMPUTESUMMARYPOLYNOMIALS(x, l, C)
    COMPUTESUMMARYPOLYNOMIALS(x, r, C)
    $G[v] \leftarrow G[l] \oplus G[r]$
  end if
  Return: $G[v]$
end function

Algorithm 2 Obtain the Shapley value vector.

function AGGREGATESHAPLEY(x, v, G)
  if v has children **then**
    $l, r \leftarrow$ children of v
    AGGREGATESHAPLEY(x, l, G)
    AGGREGATESHAPLEY(x, r, G)
  end if
  if v is not the root **then**
    e edge with v as head
    i feature of edge e
    $S[i] \leftarrow S[i] + (p_e(x) - 1)\psi(\left\lfloor \frac{G[v]}{y + p_e(x)} \right\rfloor)$
    if $\mathcal{D}(e) \neq \emptyset$ **then**
      $S[i] \leftarrow S[i] - (p_{e^\uparrow}(x) - 1)\psi(\left\lfloor \frac{G[v] \odot (y+1)^{d_{e^\uparrow} - d_e}}{y + p_{e^\uparrow}(x)} \right\rfloor)$
    end if
  end if
end function

Algorithm 3 The entire LINEARTREESHAP algorithm.

1: **function** LINEARTREESHAP(x, T_f)
2: $G \leftarrow$ an array indexed by the nodes
3: COMPUTESUMMARYPOLYNOMIALS(x, root(T_f), 1)
4: $S \leftarrow$ an array indexed by the features
5: AGGREGATESHAPLEY(x, root(T_f), G)
6: **Return:** S
7: **end function**

3.2.5 Linear TreeSHAP and complexity analysis

We can obtain an algorithm in two phases by Theorem 3.2.4. Efficiently compute the summary polynomial on each node(Algorithm 1) and then evaluate for $\varphi(f, i)$ directly(Algorithm 2). Both parts of the algorithm are straightforward, computing directly through definition and tree traversal. The final value of $S[i]$ is the desired value $\varphi(f, i)(x)$ after running Algorithm 3.

To analyze the running time, one can see each node is visited a constant number of times. The operations are polynomial addition, multiplication, division, inner product, or constant-time operations. All those polynomial operations take $O(D \log D)$ time for degree D polynomial [7]. This shows the total running time is $O(LD \log D)$.

However, we never need the coefficients of the polynomials. So, we can improve the running time by storing the summary polynomials in a better-suited form, the multipoint interpolation form. Namely, we evaluate the polynomials G on a set of predefined unique points $Y = (y_0, y_1, y_2, \cdots, y_D) \in \mathbb{R}^{D+1}$, and store $G(Y) = (G(y_0), \ldots, G(y_D))$ instead. In this form, addition, product, and division take $O(D)$ time [15]. The evaluation function $\psi(G)$ also takes $O(D)$ time but needs more explanation.

Denote $\mathcal{V}(Y) \subset \mathbb{R}^{D+1 \times D+1}$ as the Vandermonde matrix of Y, where $v_{i,j} \in \mathcal{V}(Y) = y_i^j$ is the jth power of y_i.

Lemma 3.2.5. *Let $p, q \in R[x]_d$, and its coefficients A and B, respectively, then we have*

$$\langle p, q \rangle = \langle A, B \rangle = \langle p(Y), \mathcal{V}(Y)^{-1} B \rangle.$$

Proof. Polynomial evaluation can be considered as the inner product of the coefficient and

Vandermonde matrix of input Y. Namely $p(Y) = A \cdot \mathcal{V}(Y)$. Therefore $\langle p(Y), \mathcal{V}(Y)^{-1}B\rangle = \langle A \cdot \mathcal{V}(Y), \mathcal{V}(Y)^{-1}B\rangle = \langle A, \mathcal{V}(Y) \cdot \mathcal{V}(Y)^{-1}B\rangle = \langle A, B\rangle$ completes the proof. □

In order to compute the inner products $\langle G, B_d\rangle$ in $O(D)$ time, we have to precompute $N_d = \mathcal{V}(Y)^{-1}C_d$, where C_d is the coefficient of B_d, for all $0 \leq d \leq D$. This can be done simply in $O(D)$ time for each d, so a total of $O(D^2)$ time.

By storing the polynomial in interpolation form, all our polynomial operations on each node take $O(D)$ time. Therefore, the total running time is $O(LD)$.

Besides the summary polynomials, the algorithm uses constant space to store information on nodes and edges. Each summary polynomial takes $O(D)$ space to store. Therefore, the algorithm takes $O(LD)$ space. Nevertheless, we can save space by realizing the algorithms only need a single top-down and bottom-up step. The algorithm consumes the summary polynomials on the spot by joining two steps into one. Hence, the total space usage will be bounded by $O(D)$ times the stack size, bounded by D, the depth of the tree. The final total space usage is improved to $O(D^2)$.

Remark Even though Y can be arbitrarily chosen based on the maximum depth of the tree D, it is shown that Chebyshev points are near-optimal in numerical stability [76]. In our Linear TreeShap implementation, we used the Chebyshev points of the second kind.

3.3 Experiments

Computational Platform. All experiments were run on a single workstation with an Intel Xeon (8-core) @ 2.20GHz CPU and 64GB of RAM, using Ubuntu 20.04 (64-bit) and Python 3.9. To ensure fair comparisons, we restricted the algorithms to use a single CPU core, including those (e.g., FastTreeShap) that can exploit parallel execution.

Practical Run Times. In typical use cases, we have observed that Linear TreeShap takes a few milliseconds per instance per tree when $D \leq 20$ (a typical maximum depth for tree-based models in practice). For example, on a tree with a depth of 10 and around

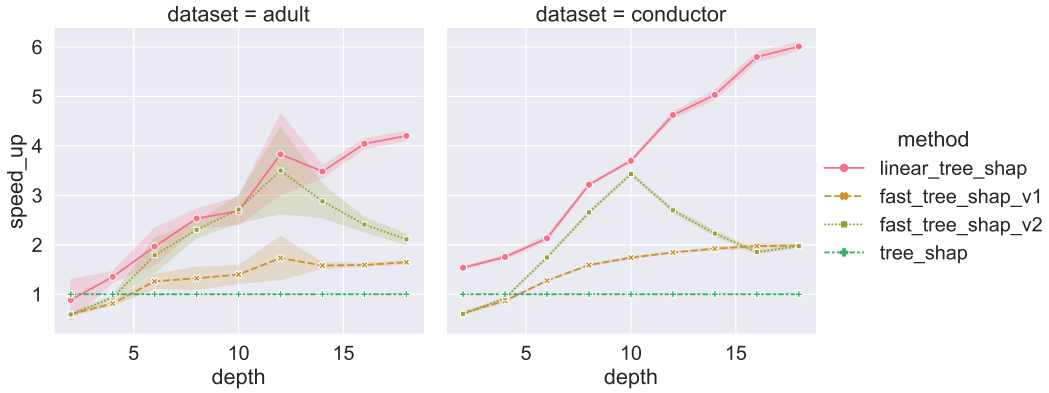

Figure 3.2 – Speed up comparison

Datasets	# Instances	# Attributes	Task	Classes
adult [36]	48,842	64	Classification	2
conductor [25]	21,263	81	Regression	-

Table 3.2 – Datasets

1,000 leaves, Linear TreeShap often runs in 2–3 milliseconds per instance, whereas the original TreeShap might require an order of magnitude more time. Thus, Linear TreeShap significantly speeds up real-world random forests or gradient-boosted trees of moderate depth without sacrificing exactness.

We run an experiment on both regression dataset adult and classification dataset conductor(summary in Table 3.2) to compare our method and two popular algorithms, TreeShap and Fast TreeShap. We explain Trees with depths ranging from 2 to 18. To align the performance across different depths of trees, we plot the ratio between the time of Tree Shap and the time of all methods in Figure 3.2. We run every algorithm on the same test set 5 times to get both average speeds up and the error bar. And for fair comparison purposes, all methods are limited to using a single core.

Linear TreeShap is the fastest among all setups. Due to heavy memory usage, Fast TreeShap V2 falls back to V1 when the tree depth reaches 18 for the dataset conductor. Since the degree of the polynomial is bounded by the tree's depth and the number of unique features per decision rule, with deeper depth, the dataset Conductor has much more speed-up gains thanks to a higher number of features. We can conclude that the Linear TreeShap

is more efficient than all state-of-the-art Shapley value computing methods in theory and practice.

Code Availability. A reference implementation of Linear TreeShap, along with Python scripts for reproducing our experiments and benchmarks, is publicly available on GitHub [1]. We encourage the community to explore, contribute, and adapt the code to a variety of tree-based models.

1. `https://github.com/yupbank/linear_tree_shap`

Chapter 4

Improved Binary Splitting of Categorical Features in Decision Tree Learning

4.1 Introduction

Decision trees are widely used for classification and regression tasks in machine learning. The CART decision tree induction method with binary splits *in* [9, 58] has become the foundation for most modern tree family algorithms. However, categorical feature splitting is still understudied despite decision trees' success and widespread use. With any (feature, label)-pair, the objective for the binary split in decision tree learning is to find the split that maximizes impurity improvement. Given a feature with K unique elements, if the feature type is numerical, $K-1$ possible two-way splits are linear in K. This is why binary split in decision learning is traditionally considered a trivial optimization problem. If the feature type is categorical, there are $2^{K-1}-1$ possible two-way splits, which is exponential in K. Then the complexity of obtaining the exact solution depends on the number of possible splits,

which is exponential.

Most popular frameworks like XGBoost [13] leave categorical feature input for the user to deal with. It treats the feature encoding problem as part of preprocessing and does not approach the issue. Another well-known framework LightGBM [33], supports numerical encoding for categorical features, as suggested in [20]. The Catboost framework [52] comes with more sophisticated options for numerical transformation by utilizing various types of target statistics. They propose to use different target statistics validated with a holdout dataset to do the numerical encoding, which is data-dependent and does not necessarily generalize well.

Most of the categorical features supported implementation use the method proposed in [20], which preprocesses categorical features by numerical encoding. The original method uses the mean target value to encode a categorical feature. It was first developed for the regression tree with the mean least squares objective. Later on, both *in* [9] and [57] generalize a similar mechanism to the 2-class classification problem. However, as *in* [57] stated, this does not extend beyond the 2-class classification problem. For more than 2-class classification, [41] proposed various approximations. It thus appears that, for binary splitting categorical feature, there is a gap between 2-class and n-class classification.

The gap between 2-class classification and more than 2-class classification motivated us. Intuitively, the binary split of categorical features in 3-class classification should not be as hard as in 30-class classification. Since the binary split of categorical features in 2-class classification has a linear subspace, The 3-class classification should also have some subspace with low-order polynomial complexity instead of exponential.

Intuition aside, the binary splitting of categorical features is a nontrivial optimization problem. For n-class classification tasks, the complexity of obtaining the exact maximum impurity improvement is exponential among existing solutions. All the previous approximated binary splitting methods were compared from a decision tree learning perspective. Classification accuracy and generalization errors were used to measure the goodness of binary splitting. We would argue that over-fitting or generalization of decision trees depends on multiple

factors, such as the tree's depth, the dataset's property, and more. On the other hand, in n-class classification task, $n \geq 2$, if binary splitting on categorical features can guarantee optimal. Generalization or overfitting of the decision tree is not different from the 2-class classification decision tree. The goodness of approximation of binary split should be studied in a much self-contained setup, optimization alone.

We propose a novel state space for the binary split to bridge the gap. The state space is based on additive sufficient statistics. Additive sufficient statistics is splitting a categorical feature with K unique values $F = \{F_1, \cdots, F_i, \cdots, F_K\}$. We use $G = \{g_1, \cdots, g_i, \cdots, g_K\} \in \mathbb{R}^n$ to capture the impurity of having single value F_i in one partition. Namely, we can get the impurity improvement of a partition having only F_1 via some function $f(g_1)$. By definition, sufficient statistics is also additive. When having both F_1, F_2 in one partition, the partition statistics is $g_1 + g_2$. And the impurity of such partition can be obtained by $f(g_1 + g_2)$.

All possible splits of F are also its power set. Thus we can embed all possible splits using a power set of sufficient statistics G. The powerset of G forms an n-dimension discrete state space

$$Z = \{\sum_{i \in S} g_i : S \subset F\} \subset \mathbb{R}^n.$$

The complexity of our state space is still exponential. Meanwhile, popular splitting criteria (e.g., Gini impurity, entropy impurity, variance reduction) are in the regime of convex objectives. The maximum of convex objectives can be obtained on the boundary of the convex domain. In our paper, we study the property of the convex hull of Z, which is central in reducing complexity towards the optimal partition. By definition, the convex hull of set Z is a very well-studied geometric object: Zonotope.

Using the property of Zonotopes, we give new proof of the linearity of computing Gini/Entropy/Variance impurity improvements for 2-class classification and regression. For more than 2-class classification, we also provide a randomized Zonotope vertex enumeration algorithm *BSplitZ*. We empirically study the performance using real-world datasets. And we limit the scope to an optimization problem.

4.2 Methodology

This section presents our state-space model on the binary splitting categorical features. Firstly, using the Gini impurity as a specific splitting criteria for 2-class classification, we demonstrate the basis of our formulation. Secondly, we present an integer programming formulation and generalize it to all convex splitting criteria. Thirdly, we prove the linearity in computing for categorical feature binary split for 2-class classification. Finally, we introduce an asymptotically optimal solution – *BSplitZ* – for the categorical feature splitting for more than 2-class classification.

4.2.1 Preliminaries

In an n-class classification setting, the dataset for the problem of binary splitting a categorical feature typically can be summarized as m (feature, (label, weight))-tuples: $(x_1,(y_1,w_1)),\cdots,(x_i,(y_i,w_i)),\cdots,(x_m,(y_m,w_m))$; where $x_i \in X, (y_i,w_i) \in Y$, $\|X\| = K$ (cardinality of the feature), $\|\{y_i : (y_i,w_i) \in Y\}\| = n$ (number of classes), and $w_i \in \mathbb{R}^+$ is the weight, being $w_i = 1$ if not specified. The goal is to find a two-way partition that maximizes some splitting criteria.

Now we use 2-class classification with Gini impurity splitting criteria, for example, $y_i \in \{0,1\}$. To find an optimal binary split, one needs to maximize the objective function: Gini impurity improvement I. Gini impurity $\mathcal{GI}(Y)$ of the dataset is defined as:

$$\mathcal{GI}(\mathcal{Y}) = 1 - \left(\frac{\sum_{i=1}^{m} w_i y_i}{\sum_{i=1}^{m} w_i}\right)^2 - \left(\frac{\sum_{i=1}^{m} w_i - w_i y_i}{\sum_{i=1}^{m} w_i}\right)^2 \tag{4.1}$$

which captures the idea that class distribution should be as different as possible in one partition. We use l, r to represent the partition index set for the left and right sides for a given split. Gini impurity improvement $I(Y,\ Y_l,\ Y_r)$ can be expressed as:

$$\sum_{i=1}^{m} w_i\ \mathcal{GI}(Y)\ - \sum_{i\in l} w_i\ \mathcal{GI}(Y_l)\ - \sum_{i\in r} w_i\ \mathcal{GI}(Y_r). \tag{4.2}$$

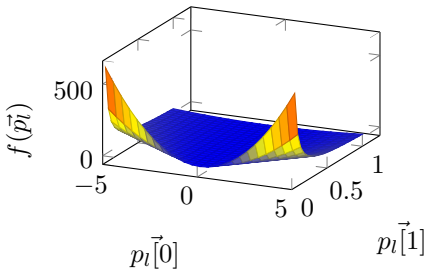

Figure 4.1 – Unnormalized Decision surface of all possible binary splits

4.2.2 Parameterizing Impurity Maximization

With some simplifications, from Eq 4.2, we have $I(Y, Y_l, Y_r)$ as:

$$\frac{\left(\sum_{i\in l} w_i y_i\right)^2}{\sum_{i\in l} w_i} + \frac{\left(\sum_{i\in r} w_i y_i\right)^2}{\sum_{i\in r} w_i} - \frac{\left(\sum_{i=1}^{m} w_i y_i\right)^2}{\sum_{i=1}^{m} w_i}. \tag{4.3}$$

By denoting $\vec{p_l} = (\sum_{i\in l} w_i y_i, \sum_{i\in l} w_i)$, $\vec{p_r} = (\sum_{i\in r} w_i y_i, \sum_{i\in r} w_i)$ and $\vec{c} = (\sum_{i=1}^{m} w_i y_i, \sum_{i=1}^{m} w_i)$, the objective function $I(Y, Y_l, Y_r)$ can be further simplified as:

$$\frac{(\overrightarrow{p_l}[0])^2}{\overrightarrow{p_l}[1]} + \frac{(\overrightarrow{p_r}[0])^2}{\overrightarrow{p_r}[1]} - \frac{(\overrightarrow{c}[0])^2}{\overrightarrow{c}[1]} \tag{4.4}$$

where $\vec{p_l}[0], \vec{p_l}[1]$ simply refer to the first and second element of $\vec{p_l}$. Since $\vec{c}$ is a constant about the dataset, and $\vec{p_r} = \vec{c} - \vec{p_l}$, we can parameterize the Gini impurity improvement with only $\vec{p_l}$:

$$I(Y, Y_l, Y_r) = f(\vec{p_l}) \tag{4.5}$$

Therefore, finding the optimal split is equivalent to finding the $\vec{p_l^*}$ that maximizes function $f(\vec{p_l})$. For example, for a dataset with $\vec{c} = (10, 100)$, we can visualize the decision surface $f(\vec{p_l})$ in Fig 4.1. The x and y axis correspond to the first and second dimensions of $\vec{p_l}$ respectively. The Gini impurity improvement is on the z axis.

4.2.3 Generalize the state space to both classification and regression splitting criteria

Both Gini and Entropy impurity are functions of class distribution. It is easy to see that $\vec{p_l}$ contains sufficient statistics to compute the Entropy impurity too. Variance reduction can adopt similar parameterization with a slight change in definition: $y_i \in \mathbb{R}$. With this parameterization, the objective of Variance reduction has a similar form to the Gini impurity. We will refer to both regression and 2-class classification as *binary task* in the remainder of this paper. And it is easy to see for binary tasks, $\vec{p_l} \in \mathbb{R}^2$, because it is sufficient to quantify the impurity of any partition with only two numbers. Meanwhile, for n-class classification, the dimension of $\vec{p_l}$ is the same of the number of classes n, specifically: $\vec{p_l} = (\sum_{i \in l_1} w_i, \cdots, \sum_{i \in l_j} w_i, \cdots \sum_{i \in l_{n-1}} w_i, \sum_{i \in l} w_i)$. Here l_j stands for the set of left-side instances with class j. To unify the terminology, we will refer to a more than 2-class classification task as *a high-dimensional task* in the remainder of this paper.

4.2.4 Constructing the Domain of Interest

Even though we can parameterize the binary split with $\vec{p_l}$ and obtain the objective function f, the domain of interest of $f : Z \to \mathbb{R}$ is only a subset of $\mathbb{R}^2$. It is constrained by all possible splits we can get from the dataset. For different features with different types, the domain of f varies. We construct the domain of interest in two steps:

1. For the feature F with cardinality K (K unique values: $\{F_1, \cdots, F_i, \cdots, F_K\}$), we get $G = \{\vec{g_1}, \ldots, \vec{g_i}, \ldots, \vec{g_K}\}$, where $\vec{g_i} = \left(\sum_{j \in F_i} w_j y_j, \sum_{j \in F_i} w_j\right)$. We use the notation for matrix and set interchangeably, with indexing across elements and rows, respectively. Namely, we slightly abuse the notation of F_i such that it is also used as an index set for data with feature value F_i.
2. Enumerate statistics for all possible splits using G. With a powerset index matrix $M \in \{0, 1\}^{(2^{K-1}-1) \times K}$, the domain of f is $Z = M \cdot G$.

To visualize the full process, we generate some random G with feature cardinality $K = 5$.

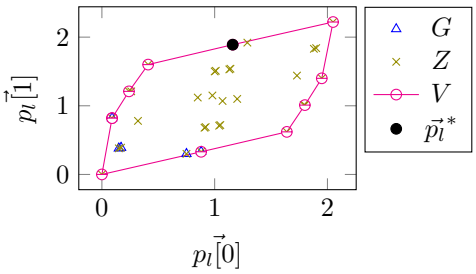

Figure 4.2 – Solution space

And enumerate all the possible splits Z using G: the order of G does not matter since Z contains all possible combinations. Recall that both G and Z are discrete sets in $\mathbb{R}^2$, so we use a scatter-plot to demonstrate G, and Z in Fig 4.2 respectively.

To summarize, the optimal splitting finding is a 0-1 integer programming problem:

$$\arg\max_i f(M_i \cdot G) \quad \text{s.t.} \quad i \in \mathbb{Z} \cap [0, 2^{K-1} - 1) \tag{4.6}$$

Recall, here M is a powerset index matrix $M \in \{0,1\}^{(2^{K-1}-1)\times K}$. We use M_i to represent the ith row of M. According to step 2, the row number of M is exponential with the feature cardinality. And in this case, brute-force enumeration would have exponential complexity. For example, for a feature with cardinality $K = 20$, the complexity is proportional to 2^{20}, which is not accessible both computationally and memory-wise. Meanwhile, it has been proven by [8] that common splitting criteria (Entropy, Gini, and Variance) are convex functions defined on $\vec{p_l}$. We can leverage the property of the convex function and the convex set: the maximum of a convex function defined on the convex domain can be obtained on the extreme points/vertices of the convex domain. The best split $\vec{p_l^*}$ (where f is maximized) stays on the convex boundary and among extreme points V. We are interested in a method that produces the extreme points of the convex hull of Z directly from G (since the optimal solution only lies on the boundary of the hull). This is a well-studied subject in geometry, called Zonotope vertex enumeration, which focuses on enumerating all vertices of a Zonotope Z. We also plot the optimal split $\vec{p_l^*}$ in Fig 4.2, which is on the convex boundary V as expected.

The domain of interest in binary splitting categorical features reduces from exponential powerset to vertex of some Zonotope. If we can efficiently generate all vertices, we can also find the optimal binary split efficiently. We can partially answer the complexity difference between 3-class- and 30-class classification problems. In a 3-class classification, we need to enumerate the vertex of a Zonotope in dimension 3. And in the 30-class classification, the Zonetope is in dimension 30. The properties of Zonotope and its vertex enumeration complexity will fulfill the gap between 2-class classification and n-class classification.

4.2.5 Zonotopes

The domain of interest Zonotope Z is a convex, centrally symmetric polytope, that is, the n-dimensional linear projection of a K-dimensional hypercube. More precisely, let $G \in \mathbb{R}^{n \times K}$ with $n < K$, the Zonotope Z is defined as

$$Z = Z(G) = \{G \cdot x : x \in [0,1]^K\} \tag{4.7}$$

The Minkowski sum of line segments P and Q is $P \bigoplus Q = \{\vec{p} + \vec{q} : \vec{p} \in P, \vec{q} \in Q\}$. The Zonotope has an equivalent representation as a Minkowski sum of K line segments in $\mathbb{R}^n$, allowing us to write:

$$Z = \{x\vec{g_1} : x \in [0,1]\} \bigoplus \cdots \{x\vec{g_i} : x \in [0,1]\} \cdots \bigoplus \{x\vec{g_K} : x \in [0,1]\}$$

where $\vec{g_i} \in \mathbb{R}^n$ is the i-th column of G. These vectors $G = \{\vec{g_1}, \ldots, \vec{g_i}, \ldots, \vec{g_K}\}$ are the *generators* of Z. The center of the Zonotope is $\frac{\sum_{i=1}^{K} \vec{g_i}}{K}$. Each vertex has a symmetric sibling with respect to the center. If G does not contain any zero vector and no two generators are scalar multiples of each other, this implies that the Zonotope is in general position. There exist $2\sum_{i=0}^{n-1} \binom{K-1}{i} = O(K^{n-1})$ vertices of a general positioned Zonotope [19]. The solution of the maximum convex function, defined on a convex set, has to be among the extreme points of this convex set, namely the vertices V. Our optimization problem can now

be solved on a subdomain $V \subset Z$. In [50], the authors give an algorithm for enumerating extreme points of Z. However, it requires exponential complexity $O(|Z| \cdot |V|) = O(2^K \cdot K^{n-1})$. It is even worse than enumerating all possible splits, which requires only $O(2^K)$.

For a categorical feature with cardinality K, in n-class classification, there are $O(K^{n-1})$ vertices from its domain of interest. Namely, binary split categorical features with 2-class classification have a subspace with $O(K)$ complexity, with 3-class has a subspace with $O(K^2)$ complexity, and with 30-class has a subspace with $O(K^{29})$ complexity. So indeed, the 3-class classification is not as complex as the 30-class. Yet, knowing the complexity is not enough; being able to enumerate them is what matters.

4.2.6 Zonotope Vertex Enumeration in Dimension Two

After showing both the domain of interest in a Zonotope and the properties of a Zonotope, we can have:

Theorem 4.2.1. *For a binary task with convex splitting criteria (e.g., Gini impurity, Entropy impurity, Variance reduction), there exists a linear subspace of splits for the maximum binary split finding of a categorical feature.*

Proof. In the 2-class classification task, all possible, sufficient statistics of binary splits from a categorical feature are enclosed in a $2d$ Zonotope. And there are $2K$ extreme points (vertices) for $2d$ Zonotope from [4]. Thus the vertices of such Zonotope are the linear subspace. □

In two dimensions, when all generators G are in the first and second quadrants, $\forall g \in G, g[1] \geq 0$, a fast algorithm enumerates all vertices of Z. It consists of sorting the generators based on their angles in increasing order and then adding them in a clockwise manner. The resulting trace forms a 2d convex closure, where all endpoints of the line segments are vertices. Since all the generators are in the first and second quadrants, $(0, 0)$ is obviously going to be a vertex. Following Jarvis's march [31] method of finding a convex hull, the rightmost point is the generator with the largest angle($arctan(g[1], g[0])$). And since all the generators

are in the first quadrant, the angle of the sum of two generators is in between them. This second-largest vertex is the sum of the largest and second-largest angle generators. Thus we can recursively find the rest of the vertices by following the order by angle. There is also another proof provided by Property 9 in [44].

Thus encoding the categorical feature with target statistics by [20] is equivalent to the special case 2d Zonotope vertex enumeration algorithm (generators in the first and second quadrant). Since all the vertex of the convex hull is an optimal subspace for the exponential space, such encoding is optimal. This encoding can be extended to data with weights, gradients, or hessian for convex objectives such as Gini, Entropy impurity, variance reduction, or any convex combination.

Meanwhile, such a short-cut algorithm is not generally applicable to 2d Zonotope vertex enumeration. Namely, we need to ensure the weights are positive in every instance. Recall the second component. A generator is the sum of weights, $\vec{g}_i = \left(\sum_{j \in F_i} w_j y_j, \sum_{j \in F_i} w_j\right)$. With negative weights, the popular algorithm cannot produce the corresponding vertex. Negative weights might not be common, but evaluated hessian are used as weights in the gradient boosting algorithm. And they can be negative.

4.2.7 Zonotope Vertex Enumeration in High Dimensions

For a high-dimensional task, the simple algorithm for the binary task is not suitable. There are more advanced methods, such as reverse search, that provide the time complexity $O(n \cdot K \cdot LP(K, n)|V|)$ [4]. Here $LP(K, n)$ stands for the complexity of linear programming with K variables, and n constraints $|V|$ stands for the number of vertices. The size of the set of vertices $|V|$ dominates its complexity, yielding $O(K^{n-1})$. Collecting all vertices would require exponential complexity, making it nonideal. Conveniently, there is an algorithm using randomized enumeration proposed in [70]. We adopted the idea within our method: *BSplitZ*. The vertex enumeration is shown in Algorithm 4.

By setting a fixed number of samples, one can sample potential optimal splits from V in a controllable manner. The probability of obtaining a given sample vertex is proportional

Algorithm 4 Randomized algorithm to enumerate the vertices of the Zonotope

Input: Empty set V and generators G
Parameter: number of samples s
Output: V

1: $k = 2 \cdot \sum_{i=0}^{n-1} \binom{K-1}{i}$
2: **while** $|V| < k$ or $s > 0$ **do**
3: Draw $\vec{x}$ independently from 0 centered, I covariance Gaussian in $\mathbb{R}^n$
4: Compute $\vec{v} = G \cdot sign(G^T \cdot \vec{x})$
5: Put $\vec{v}$ and $-\vec{v}$ into V
6: Set $s = s - 1$
7: **end while**
8: **return** V

to the angle of the *normal cone* at that vertex over the normal fan (the union of all normal cones) of the Zonotope. In high dimensional space, the angle of a normal cone can be extended to the solid angle enclosed by the normal vectors of facets.

The probability of sampling any given vertex is given by the cumulative density function of a degenerated zero mean, multivariate normal distribution evaluated at the given vertex. For example, the probability of obtaining vertex $(0, 0, \ldots, 0)$ is the orthant probability: $P(X < 0), X \sim N(0, G \cdot G^T)$. This degenerated distribution guarantees that not all 2^K samples are possible or have a probability greater than 0. The cumulative density function of arbitrary dimension multi-variate normal distribution does not have an analytical form. To achieve an optimal result, we have to enumerate all vertices. We also put effort into determining the expected number of samples required to enumerate all vertices. We can treat every vertex as a coupon. Sampling the Zonotope vertex until we visited all the vertices is analogous to drawing coupons until we have collected all the coupons. Calculating the expected number of draws/samples required to collect/enumerate all coupons/vertex is another well-studied problem, the nonuniform coupon collection problem. Different coupon/vertex have different weights that correspond to the orthant probability. Based on Theorem 3.5 from [67], the expected number of draws is bounded by $O(\beta \cdot |V| \cdot \log(|V|))$ and $O(\frac{1}{\beta} \cdot |V| \cdot \log(|V|))$, where β is the ratio of angles between the minimum angle normal cone and the maximum angle normal cone, and recall $|V| = \sum_{i=0}^{n-1} \binom{K-1}{i} = O(K^{n-1})$ in our case. In practice, the number of draws is usually fixed as a constant. Therefore, the running time complexity is low.

Feature cardinality	Number of classes	
	3-8	10-59
2-10	52	16

Table 4.1 – Number of optimization problems grouped by the range of feature cardinality and the number of classes.

4.3 Experimental Evaluation

Without making further data assumptions, we use various real-world datasets from OpenML [1] to study the effectiveness and efficiency of BSplitZ in practice. All classification datasets on the Openml platform containing more than 10k instances, three classes, and one categorical feature are used. The datasets are tami with id 1505, kropt with id 184, settle_crime with id 41960, pokerhand with id 155, nursery with id 1568, kddcup with id 1113, fars with id 40672 and dibetes with id 4541. We first focus on the optimization problem alone and compare the optimization ranking of BSplitz with other methods. And then, we pick a subset of feasible feature encoding methods to compare in a standard machine learning setting.

Computational Platform. All experiments presented in this chapter were conducted on a workstation equipped with an Intel Xeon CPU (8-core) @ 2.20GHz and 64GB of RAM, running Ubuntu 20.04 (64-bit) with Python 3.9. We ensured all baseline methods and our method ran in a single-threaded mode unless otherwise specified.

We extract all the categorical features in those data sets and form 68 optimization problems. 11 categorical features with more than ten values are omitted from the comparison because we need to include brute force in the baseline. We summarize the problems' distribution in Table 4.1.

We run a comparison with both a collection of state-of-the-art categorical feature encoding methods such as M-probability estimate of likelihood(m), James-Stein estimator(james), CatBoost coding(cat), Target encoding(target), Encodes categorical features as ordinal(ordinal),

1. `https://www.openml.org/`

Count encoding(counts), Base-N encoder(base), Binary encoding(binary) (from [2]) and direct splitting methods such as One hot encoding(one hot), One-Versus-All by Class(one vs all), Principal Component-Based Partitioning based(pca), Pull Left by Purity based methods(pull left)(from [3]) using both Gini and Entropy impurity improvement.

Underlying Model. Throughout our experiments and comparisons, we primarily used CART trees as the underlying model to evaluate the practical performance of our proposed splitting approaches. This choice aligns with the observation that encoding the categorical feature with target statistics [20] is effectively a special case of the 2d Zonotope vertex enumeration in the context of CART trees.

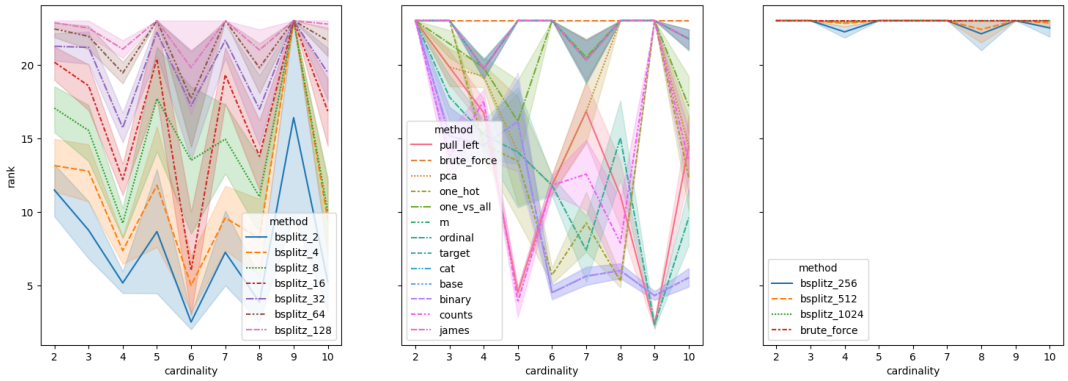

Figure 4.3 – Ranking of Gini impurity improvement for all 65 optimization problems. Where the ranks higher, the better.

Using all the methods, we rank the Entropy and Gini impurity improvement for every (feature, label) pair. Since some methods are stochastic, ten rounds of impurity reduction rank comparison are conducted to estimate the standard deviation. The maximum rank is used. Thus the higher rank, the better the model. As 23 methods are used in our comparison, the best model will rank 23, which is also brute force.

Integer Programming as a Baseline. We also considered directly solving the associated 0-1 integer programming (IP) problem to find the optimal split as another potential

2. https://contrib.scikit-learn.org/category_encoders/
3. https://www.mathworks.com/help/stats/splitting-categorical-predictors-for-multiclass-classification.html

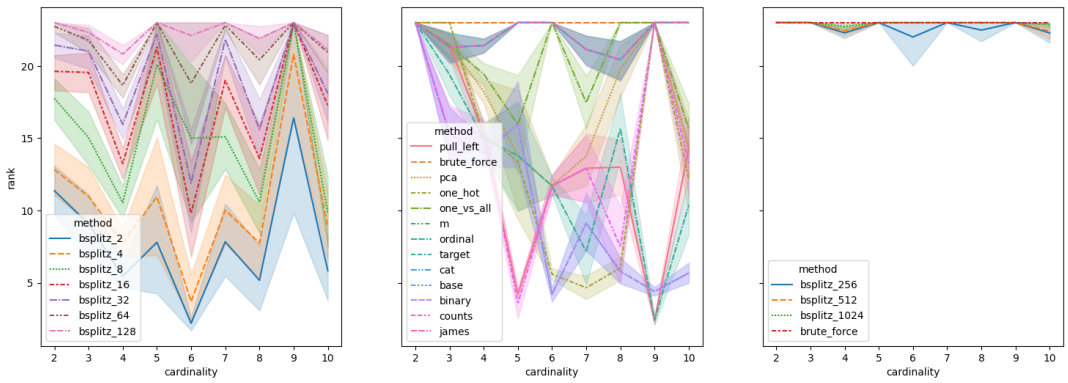

Figure 4.4 – Ranking of Entropy impurity improvement for all 65 optimization problems. Where the ranks higher, the better.

baseline. However, we found that generic IP solvers became intractable for even moderate feature cardinalities (e.g., $K > 15$), due to the exponential number of possible partitions. Thus, while IP could serve as an exact baseline on very small categorical features, our focus remained on randomized and other heuristic methods for larger K, which aligns with the scope of bridging exponential complexity in practice.

The number of samples for BSplitZ is also explored in our experiment. We add a suffix of the number of samples to BSplitZ in the method name. We report both Gini and Entropy result in Fig 4.3 and Fig 4.4. As expected, as we increase the number of samples for BSplitZ, the ranking performance also improves. BSplitZ outperforms all other methods(except for brute force) with around 256 samples.

And we also study the efficiency of different methods. As shown in Fig 4.5 Only a few encoding-based methods(base, binary) have significantly slower speeds. And the brute force method is the second slowest, as expected. The rest of the methods have more or less similar speeds.

We also conduct a second experiment with standard train/test accuracy evaluation to assess the practical impact on machine learning tasks. We implement BSplitZ as a feature encoding mechanism to compare with other popular feature encoding methods.

As all the features and limitations of the available implementation of encoding methods are used in the previously mentioned datasets, we only compare nine methods for accuracy.

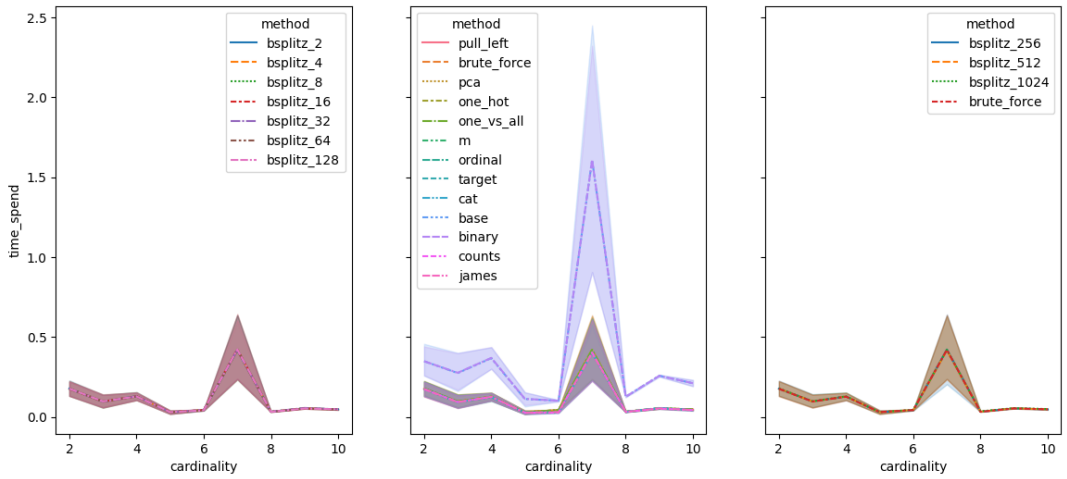

Figure 4.5 – Speed comparison of different methods in seconds over different feature cardinality

In this comparison, we also set a fixed sample size of 256 for BSplitZ. Ten rounds of accuracy ranking are performed. We first split the dataset into train/test(7:3 ratio) folds in every round. Then categorical features are encoded into numerical. We also vary the depth of trees in 1, 5, 10, and 15. We rank the accuracy obtained in each setting across all tree depths, then aggregate them and report in Figure 4.6 and Figure 4.7. Similarly, we use max rank in comparison. Thus the higher method ranks, the better the encoding method. And the best rank is 9.

BSplitZ is ranked among the top 3 across all datasets. And particular, the BSplitZ method outperforms the rest in training and testing accuracy on large datasets.

4.4 Conclusions

We proposed a new general framework for binary decision tree splitting. We generalized the framework to process the categorical features directly. And we give further insight into the binary split in the decision tree and add new proof to some empirical encoding methods. We also presented BSplitZ, an asymptotic optimal algorithm, for categorical feature splitting within this new framework. We evaluated the new algorithm across several different datasets. Our novel methodology outperforms other state-of-art methods.

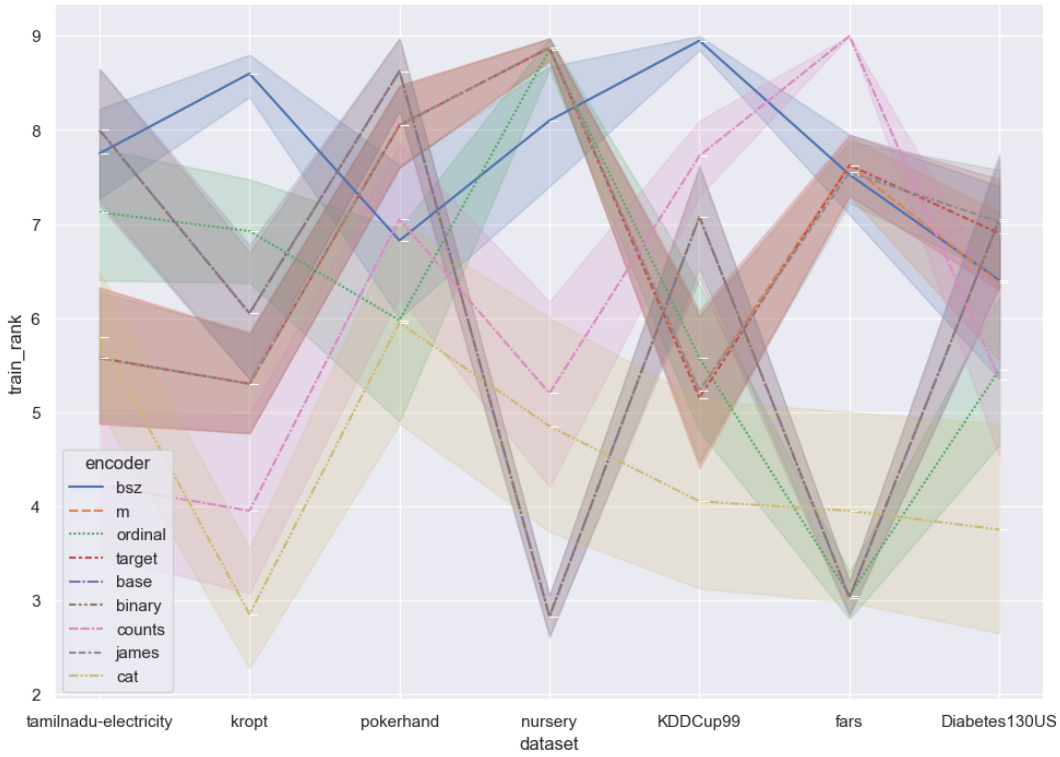

Figure 4.6 – We rank the training accuracy for different methods, the higher, the better

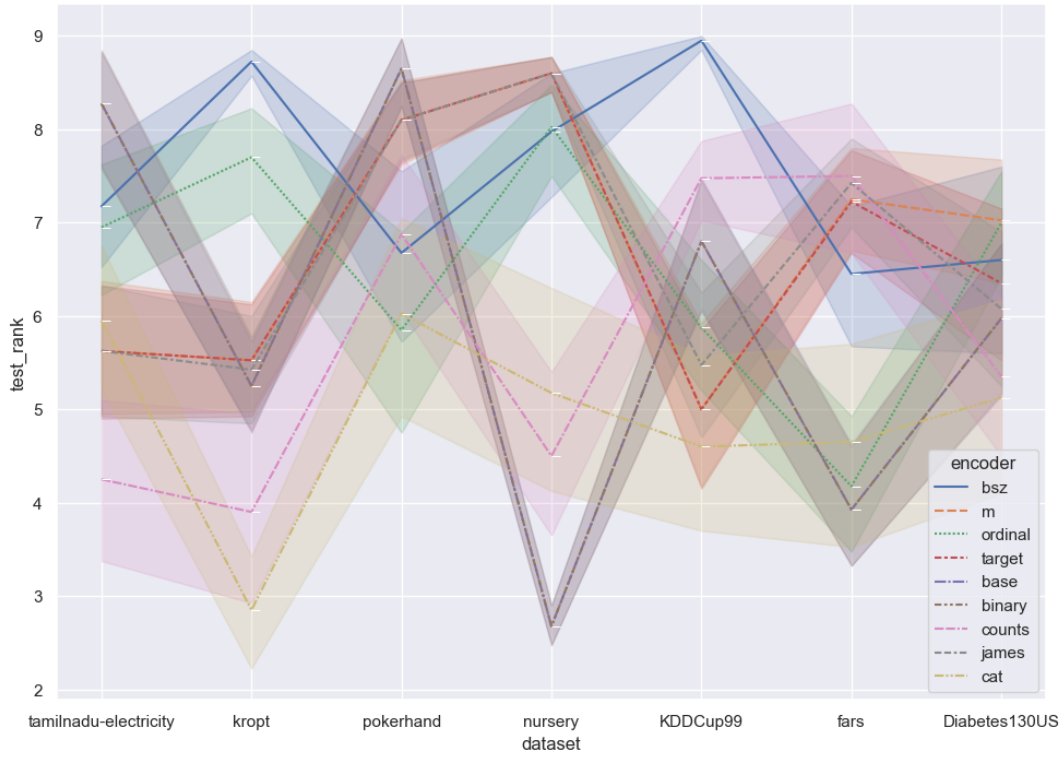

Figure 4.7 – We rank the testing accuracy for different methods, the higher, the better

Chapter 5

Binary Split Categorical feature with Mean Absolute Error Criteria in CART

5.1 Introduction

The CART family of algorithms (random forest, gradient boosting tree) is well-known for its top performance on tabular data. Real-world tabular data often contains not only numerical but also categorical features. The CART algorithm recursively partitions the input dataset with a binary split optimization step and terminates when reaching a minimum number of instances. While traditional machine learning models only work with numerical data, the CART family of algorithms can process categorical features directly. This flexibility is because the binary split optimization step in the CART algorithm only requires feature data types that allow different subsets.

The binary split step is recognized as a major bottleneck regarding the computational efficiency of tree learning algorithms [12]. Specifically, when processing categorical fea-

tures, the associated discrete set topology can result in an exponential search space for binary splits. As a result, various numerical encoding methods have been developed to address this limitation. Consequently, many popular tree-based machine learning software packages (such as XGBoost [13], LightGBM [33], and Catboost [52]) only support numerical data or include automatic numerical encoding methods for categorical data. On the other hand, only a subset of splitting criteria (such as mean squared error and Gini impurity) have optimally guaranteed numerical encodings for categorical data [27]. The splitting criterion MAE (Mean Absolute Error) lacks a proven optimal numerical encoding. MAE is more robust when dealing with outliers and skewed distributions and is widely adopted in various statistical domains. This criterion's most successful and practical numerical encoding method is median-based heuristic numerical encoding. This highly efficient heuristic takes $O(n \log n)$ time, where n is the dataset size. However, the heuristic may not be optimal for large, randomly generated examples. Nonetheless, a simple proof of its non-optimality remained elusive, as verifying the optimality of large random examples by hand is impractical [73].

To address this, we first prove there do not exist optimal numerical encoding. While the proof itself is relatively straightforward, the significance of this result is reflected in the substantial effort invested in seeking the optimal numerical encoding method. For instance, dozens of unsupervised numerical encoding methods are under development [45]. And then we introduce our new algorithm, which may also hold independent interest as it efficiently solves the unimodal cost 2-median problem.

5.2 Preliminaries

In regression tree learning, during a node split computation, the goal is to find a binary partition S, S^c of $Y \in \mathbb{R}^n$, based on some feature X. Here, S^c is the complement of S with respect to Y. When X is categorical, the goal can be further simplified as finding a binary partition of $\mathcal{Y} = \{Y_1, Y_2, \dots Y_k\}$, where $Y_i \subseteq \mathbb{R}$ is the subset of the target data points with category i in feature X.

Depending on the splitting criteria, the partition must maximize or minimize an objective function. For a given set $\mathcal{S} \subseteq \mathcal{Y}$, the MAE is defined as

$$\textbf{MAE}(\mathcal{S}) = \sum_{x \in \cup \mathcal{S}} |x - \textbf{M}(\cup \mathcal{S})|. \tag{5.1}$$

Here, $\textbf{M}(\cdot)$ represents the median function, and $|\cdot|$ denotes the absolute value. When using MAE as the splitting criterion, we define the objective value as:

$$\lambda := \min_{\mathcal{S} \subset \mathcal{Y}, \mathcal{S} \neq \emptyset} \textbf{MAE}(\mathcal{S}) + \textbf{MAE}(\mathcal{S}^c). \tag{5.2}$$

The problem of computing the split S, S^c that achieves the minimum is referred to as the *MAE split problem*.

To solve for the minimum, one can enumerate all subsets of $\mathcal{Y}$, resulting in an undesirable $O(2^k)$ search space. On the other hand, the community has developed numerous heuristic numerical encoding methods.

Unsupervised Numerical Encoding Rather than enumerating all subsets, one can employ a numeric encoding/set function $e : 2^{\mathbb{R}} \to \mathbb{R}$ to establish an ordering $\preceq_e$ and select sets based on this ordering.

A numerical encoding function is considered unsupervised if specified independently of the data, meaning it is determined without observing the input.

The numerical encoding provides a natural ordering over the elements of $\mathcal{Y}$, where $A \preceq_e B$ if $e(A) \leq e(B)$. We define the *downward closed sets* of $\mathcal{Y}$ as $D'(\mathcal{Y}, e)$, which consists of sets of the form $\{A \mid e(A) \leq x, A \in \mathcal{Y}\}$ for some x.

Numerical encoding has been used as a heuristic to identify the optimal partition within the downward closed sets. Specifically, let $D(\mathcal{Y}, e) = D'(\mathcal{Y}, e) \setminus \{\mathcal{Y}, \emptyset\}$, and our goal is to find $\mathcal{S} \in D(\mathcal{Y}, e)$ that minimizes λ. This modified problem leads to a more favorable $O(k)$ search space, though optimality is not guaranteed.

For instance, the median heuristic employs the encoding function $e = \textbf{M}$ by arranging

sets based on their medians. It then enumerates the downward closed sets and their complements as potential splits. Consequently, this approach yields only $k-1$ potential splits.

Piecewise-linear functions We also introduce a few useful functions and their properties. A function $f : \mathbb{R} \to \mathbb{R}$ is *unimodal*, if there exists a c such that for any $x \leq y \leq c$, we have $f(x) \geq f(y)$, and for $c \leq x \leq y$, $f(x) \leq f(y)$.

A function $h : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is *totally monotone*, if for any $x_1 \leq x_2 \leq y_1 \leq y_2$, $h(x_1, y_1) \geq h(x_1, y_2)$ then $h(x_2, y_1) \geq h(x_2, y_2)$. A positive linear combination of totally monotone functions is totally monotone. A matrix is totally monotone if the function $h(i, j) = M_{i,j}$ is totally monotone. The main property of a totally monotone matrix is the index for row minimum is non-decreasing. That is, if M_{i,a_i} is the minimum value of the ith row, then we have $a_1 \leq a_2 \leq \ldots \leq a_n$ [51].

We use $B(f)$ to denote the set of breakpoints for a piecewise linear function. Recall the median function $\mathbf{M}(S)$ is an element $y \in S$, such that $\sum_{x \in S} |x - y|$ is minimized.

5.3 No unsupervised optimal numerical encoding for MAE

Is there a numerical encoding that can be used to find the optimal binary split for MAE? Specifically, is there an encoding function e such that the following equality holds: λ is equal to $\min_{\mathcal{S} \in D(\mathcal{Y},e)} \textbf{MAE}(\mathcal{S}) + \textbf{MAE}(\mathcal{S}^c)$?

Even though target mean-based numerical encoding for categorical features has been proven optimal in decision tree regression with mean squared error (MSE) in [9], the same heuristic does not work with MAE. Empirically, it has been shown that the median numerical encoding works most of the time for MAE splitting criteria [73]. The conclusion is made based on experiments on limited datasets. The median encoding result in sub-optimal splits was only observed through some rare, randomly generated datasets.

Still, this does not rule out the existence of other unsupervised numerical encodings that work for MAE. We prove there does not exist any unsupervised optimal numerical encoding.

Suppose a unique optimal partition of a dataset minimizes the MAE. In that case, any

encoding that works for MAE must have the encoding of all elements in one partition strictly smaller or greater than the encoding of the other partition. Formally, let $\{\mathcal{A}, \mathcal{B}\}$ forms the unique optimum partition of dataset $\mathcal{Y}$, and e is a encoding that *works* for MAE, then either $e(A) < e(B)$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$, or $e(A) > e(B)$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$. If the encoding e works for MAE, then the optimal partition is in $D(\mathcal{Y}, e)$.

Theorem 5.3.1. *No numerical encoding function works for binary split with MAE splitting criteria.*

Proof. Assume such encoding e exists and prove by contradiction via constructing a counter-example. Let $\epsilon > 0$ be some small and fixed value, say 0.01.

Let $a_1 = 0, a_2 = 2, a_3 = 3, a_4 = 5$. We define $A_i = \{a_i - \epsilon, a_i, a_i + \epsilon\}$, $A'_i = \{a_i - \epsilon, a_i + \epsilon, a_1\}$ if $i \in \{3, 4\}$, otherwise $A'_i = \{a_i - \epsilon, a_i + \epsilon, a_4\}$.

1. The unique optimum partition of $\{A_1, A'_1, A_4, A'_4\}$ is $\{A_1, A'_1\}, \{A_4, A'_4\}$, without loss of generality, let $e(A_1) < e(A_4)$.
2. The unique optimum partition of $\{A_2, A'_1, A_3, A'_4\}$ is $\{A_2, A'_1\}, \{A_3, A'_4\}$, hence $e(A'_1) < e(A'_4)$.
3. The unique optimum partition of $\{A_2, A'_2, A_3, A'_3\}$ is $\{A_2, A'_2\}, \{A_3, A'_3\}$, hence $e(A_2) < e(A_3)$.
4. The unique optimum partition of $\{A_1, A'_2, A'_3, A_4\}$ is $\{A_1, A'_3\}, \{A'_2, A_4\}$, hence $e(A_4) < e(A_1)$, a contradiction.

□

5.4 Methodology

Knowing no unsupervised numerical encoding e works for MAE, there might still be efficient algorithms that don't use any encoding. In the following section, we propose such an algorithm.

We first transform the MAE split problem into a more manageable version, which avoids the median function.

Proposition 5.4.1. *Let $\mathcal{Y}$ be a family of disjoint sets, and define*

$$\textbf{\textit{MAE}}(\mathcal{S}) \;=\; \min_{a\in\mathbb{R}} \sum_{i\in\bigcup \mathcal{S}} |i-a|$$

for any nonempty $\mathcal{S}\subseteq\mathcal{Y}$. Then

$$\lambda \;=\; \min_{\emptyset\subsetneq\mathcal{S}\subsetneq\mathcal{Y}} \big(\textbf{\textit{MAE}}(\mathcal{S}) \;+\; \textbf{\textit{MAE}}(\mathcal{S}^c)\big)$$

can be equivalently written as

$$\lambda \;=\; \min_{a,b\,\in\,\mathbb{R}} \sum_{S\,\in\,\mathcal{Y}} \min\Big(\sum_{i\,\in\,S} |i-a|,\; \sum_{j\,\in\,S} |j-b|\Big).$$

Proof. By definition,

$$\textbf{MAE}(\mathcal{S}) \;=\; \min_{a\,\in\,\mathbb{R}} \sum_{i\in\bigcup \mathcal{S}} |i-a|.$$

Hence, for any subset $\mathcal{S}\subseteq\mathcal{Y}$ with complement $\mathcal{S}^c$, we have

$$\textbf{MAE}(\mathcal{S}) + \textbf{MAE}(\mathcal{S}^c) \;=\; \min_{a\in\mathbb{R}} \sum_{i\in\bigcup \mathcal{S}} |i-a| \;+\; \min_{b\in\mathbb{R}} \sum_{j\in\bigcup \mathcal{S}^c} |j-b|.$$

Note that $\bigcup\mathcal{S}$ and $\bigcup\mathcal{S}^c$ partition all elements in $\bigcup\mathcal{Y}$. We may reorganize the sums *set by set* (i.e., over each $S\in\mathcal{Y}$), introducing two real parameters a and b. Gathering the terms for each S and observing that the choice

$$\min\Big(\sum_{i\in S} |i-a|,\; \sum_{j\in S} |j-b|\Big)$$

corresponds to placing S into $\mathcal{S}$ or its complement $\mathcal{S}^c$, respectively, yields

$$\min_{\emptyset\subsetneq\mathcal{S}\subsetneq\mathcal{Y}} \big(\textbf{MAE}(\mathcal{S}) + \textbf{MAE}(\mathcal{S}^c)\big) \;=\; \min_{a,b\,\in\,\mathbb{R}} \sum_{S\,\in\,\mathcal{Y}} \min\Big(\sum_{i\in S} |i-a|,\; \sum_{j\in S} |j-b|\Big).$$

This confirms the claimed equivalence, completing the proof. □

After the transformation, we consider the following optimization problem.

Problem 1 (Median split problem)**.** *Given $\mathcal{Y}$, a family of subsets of $\mathbb{R}$, find $a, b \in \mathbb{R}$, such that $\sum_{S\in\mathcal{Y}} \min(\sum_{i\in S} |i-a|, \sum_{j\in S} |j-b|)$ is minimized.*

Define $f_S(x) = \sum_{i\in S} |i-x|$ and $h_S(x,y) = \min\{f_S(x), f_S(y)\}$. Let $c = \mathbf{M}(S)$, when $x < c$, f_S is monotonically decreasing and when $x > c$, f_S is monotonically increasing. Hence, f_S is a unimodal function. Observe that we try to optimize $h(a,b) = \sum_{S\in\mathcal{Y}} h_S(a,b)$. The function is piecewise-linear convex, and the optimum can be achieved when a and b are breakpoints of the function.

One can compute $h(a,b)$ for each a and b and pick the minimum. There are $O(n^2)$ points to evaluate. Evaluating h takes $O(n)$ time; hence the total running time is $O(n^3)$.

However, we note that each f_S is a piecewise-linear convex function. We use this property to design a slightly faster algorithm. To this end, we introduce a much more general problem.

Problem 2 (Unimodal Cost 2-Median)**.** *Let $f_1, \ldots, f_k : \mathbb{R} \to \mathbb{R}$ be k piecewise-linear unimodal functions with a total of n breakpoints. Let $g(a,b) = \sum_{i=1}^{k} \min\{f_i(a), f_i(b)\}$. Find $a, b \in \mathbb{R}$ such that it minimizes g.*

Theorem 5.4.2. *Problem 1 reduces to Problem 2 in linear time.*

Proof. Let $f_S = \sum_{i\in S} |i-x|$. Let the input to the Problem 1 be $\mathcal{Y} = \{S_1, \ldots, S_k\}$. This reduces to Problem 2 with input functions $f_{S_1}, \ldots, f_{S_k}$. □

Therefore, we shift gear and try to solve Problem 2 in the remainder of the paper.

5.5 Algorithm for Unimodal Cost 2-Median

For a piecewise-linear function f of n breakpoints, the representation consists of the sorted list of breakpoints $x_1, \ldots, x_n$, and the corresponding values $f(x_1), \ldots, f(x_n)$. Additionally, the initial slope and the final slope are also stored. Given i, one can find x_i, and

evaluate f, the right slope of f, and the change of slope of f at x_i, all in $O(1)$ time. If we are interested in finding the value of $f(x)$ by giving x instead of any index, it takes $O(\log n)$ time by doing a binary search over the list and then interpolating adjacent breakpoints.

5.5.1 Properties of the problem

Let $f_1, \ldots, f_k$ be the input of Problem 2, and $g(x, y) = \sum_{i=1}^{k} \min\{f_i(x), f_i(y)\}$. Evaluate g for all x, y takes $O(k)$ time each if we look at x, y in order. Hence, Problem 2 has an $O(n^2 k)$ time algorithm.

This algorithm is extremely naive, looking through all possible input pairs. One might guess that if we fix a, then the function $g_a(b) = g(a, b)$ is a unimodal function, and then a binary search like procedure can be applied to find the minimum b for g_a. Unfortunately, this is false; g_a can have many local minimums. Fortunately, g is a totally monotone function.

Theorem 5.5.1. *Let $f : \mathbb{R} \to \mathbb{R}$ be an unimodal function. The function $g : \mathbb{R}^2 \to \mathbb{R}$ defined as $g(x, y) = \min(f(x), f(y))$ if $x \leq y$, and $g(x, y) = \infty$ if $x > y$, is a totally monotone function.*

Proof. Consider for any $x_1 \leq x_2$ and $y_1 \leq y_2$.

In order to show h is totally monotone, we have to show that if $\min(f(x_1), f(y_1)) \leq \min(f(x_1), f(y_2))$, then $\min(f(x_2), f(y_1)) \leq \min(f(x_2), f(y_2))$.

If we do not have $x_1 \leq x_2 \leq y_1 \leq y_2$, we will obtain an infinity case, and one can see the inequalities hold.

Hence we assume $x_1 \leq x_2 \leq y_1 \leq y_2$.

There are two cases.

Case 1. If $f(y_1) \leq f(y_2)$, then $\min\{f(x_2), f(y_1)\} \leq \min\{f(x_2), f(y_2)\}$.

Case 2. Otherwise, assume $f(y_1) > f(y_2)$. Because $f(y_1) > f(y_2)$ but $y_1 \leq y_2$, so we must have y_1 is in the decreasing part of the function f. Therefore, we must have $f(x_1) \geq f(x_2) \geq f(y_1) > f(y_2)$. Hence, $f(y_1) = \min\{f(x_1), f(y_1)\} \leq \min\{f(x_1), f(y_2)\} = f(y_2) < f(y_1)$, a contradiction. □

By Theorem 5.5.1, and the fact sum of totally monotone function is totally monotone [51],

the function we try to optimize in Problem 2 is a totally monotone function. Let $x_1, \ldots, x_n$ be all the breakpoints of all the functions ordered from small to largest. Consider the matrix M, such that $M_{i,j} = g(x_i, x_j)$, where x_i is the ith breakpoint of g, then M is a totally monotone matrix. It is useful for us to consider the index-based version of the breakpoint instead of the breakpoint itself.

For a totally monotone matrix M, the SMAWK algorithm finds the row minima of each row of M in $O(n)$ evaluations of entries in M [1]. Each evaluation takes $O(k \log k)$ time. Therefore, we obtain a $O(nk \log k)$ time algorithm.

On the other hand, the median encoding method performs k evaluations. If the same search type is performed, the median encoding method will exhibit a time complexity of $O(k^2 \log k)$. But each search does not have to be independent. In practice, by arranging the order of evaluation with better data structure, $O(n \log n)$ time complexity is achieved. Drawing inspiration from this, the subsequent section presents an accelerated algorithm that marginally increases the evaluation count while enhancing individual evaluation speed.

5.5.2 Slowing down to speed up

Recall that we are interested in finding the x and y such that $g(x, y)$ is minimized, where x, y is from a lattice grid and $g(x, y)$ evaluated on the grid results in a totally monotone matrix. We can make the evaluation dependent on predecessors through an alternative divide-and-conquer algorithm other than the SMAWK. This new divide-and-conquer algorithm will take a total of $O(n \log n)$ evaluations of the matrix, so a *slow down* in the number of evaluations. However, a total *speed up* is obtained by speeding up each individual evaluation.

We outline the idea as follows: for a fixed i, let j be the value that minimizes $M_{i,j}$. The optimum of the entire matrix must be either $M_{i,j}$, or of the form $M_{i',j'}$ where $i' < i$ and $j' \leq j$, or $i' > i$ and $j' \geq j$ [51]. Hence, this gives us a natural divide-and-conquer algorithm: find the row minimum of the center row and recursively solve the new problem on the two smaller matrices. Observe the total number of evaluations of a $n \times m$ matrix would be $T(n, m) = T(n/2, m_1) + T(n/2, m_2) + O(m) = O(m \log n)$. Since, in our case, $n = m$, we

get an algorithm taking $O(n \log n)$ evaluations.

Naively, this would give us an $O(nk \log n \log k)$ time algorithm, which is even worse than the SMAWK algorithm. The realization is this form actually helps us reason about the result in the following sense. Instead of the number of evaluations, we can show that the *running time* follows a similar recursion, and thus we obtain a $O(n \log n + k \log k \log n)$ time algorithm.

Naturally, we have to describe how to solve the two parts of the problem: Finding the row minimum and divide-and-conquer.

5.5.3 Find the minimum over a single row

Finding a minimum of a given row in the matrix M, is equivalent to answer the following question: Given functions $f_1, \ldots, f_k$, and an fixed index a, how to find $\min_b \sum_{j=1}^{k} \min(f_j(x_a), f_j(x_b))$ quickly?

For a fixed a, define the *active set* at index b to be the set of function indices j, such that $f_j(x_a) > f_j(x_b)$. Let $A_1, \ldots, A_n$ be the sequence of active sets at $1, \ldots, n$, respectively. Each function f_j moves out of the active set only once. That is, for an index $j \in [k]$, there exists an index q_j, such that for each $i \geq q_j$, we have $j \in A_i$, and $j \notin A_i$ otherwise.

Define $f_A = \sum_{i \in A} f_i$. If we can quickly evaluate f_A, and update A, then we can quickly evaluate g. Indeed, in order to evaluate $g(x_a, x_b)$, the idea is to break it down into evaluating $f_{\bar{A}}(x_a) + f_A(x_b)$ where A is the active set at index b.

It's not hard to describe a data structure that maintains the value of $f_A(x_a)$ under updates of A and a; this is precisely the evaluation data structure in Theorem 5.5.4. However, we must also decide which function has to be added or removed from A. This is done through a useful transformation. Let f be an unimodal function with the local minimum at c, define $f^\dagger : \mathbb{R} \to \mathbb{R}$ to be $f^\dagger(x) = \max\{x' \mid f(x') \leq f(x)\}$ if $x \in (-\infty, c]$ and $-\infty$ otherwise. Intuitively, this means for any $x \leq c$, if we have $x \leq x' \leq f^\dagger(x)$, then we know $f(x') \leq f(x)$. Namely, one can quickly observe if $f(x) \leq f(x')$ by checking if $x' \leq f^\dagger(x)$. See Figure 5.1.

If f is a piecewise-linear unimodal function of n breakpoints, then $f^\dagger$ is a piecewise-linear decreasing function in $(-\infty, c]$ of n breakpoints and can be computed from f in $O(n)$ time.

We can find the value of $f^\dagger(x)$ in $O(\log n)$ time. Since $f^\dagger$ can be computed in linear time when f is created, we always assume that $f^\dagger$ is computed when we use f.

Knowing $f_i^\dagger(x_a)$ for each i, then we know precisely when i moves out of the active set: i moves out of the active set when $x_b > f_i^\dagger(x_a)$ for the first time. See Algorithm 5 for implementation of ROWMINIMA.

Theorem 5.5.2. *If the input is k functions with a total of n breakpoints, row minima can be found in $O(n + k \log k)$ time.*

Proof. We analyze the running time of ROWMINIMA. Ordering the functions by evaluation of $f_j^\dagger(x_a)$ for each j, which takes $\sum_{j=1}^{k} n_j \log n_j = O(k \log \frac{n}{k})$ time, where n_j is the number of breakpoints of f_j. Sorting the k functions by their $\dagger$ value takes $O(k \log k)$ time. The linear scan takes $O(n)$ time. Hence, going through each breakpoint takes $O(n)$ time. Note $k = O(n)$, hence $O(k \log \frac{n}{k}) = O(n)$. The total running time of RowMinima is $O(n + k \log k)$. □

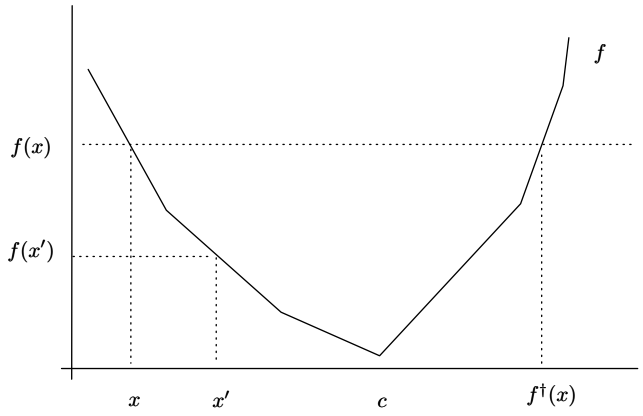

Figure 5.1 – Intuition of the † transform.

5.5.4 Divide-and-conquer

In the divide-and-conquer step, we split the problem into evaluations over 2 smaller submatrices, which are almost disjoint.

Observe that once we are searching for the optimum in a submatrix where the row index ranges from a_{min} to a_{max} and the column index ranges from b_{min} to b_{max}. All the values of

the functions outside this range are irrelevant. Hence we can safely assume we process all the functions passed into the recursive call by removing the breakpoints outside the ranges. In practice, this is not explicit but done through implicit bookkeeping. This allows us to bound the running time related to traversing the functions by $O((b_{max}-b_{min})+(a_{max}-a_{min}))$ inside each recursive call. See the recursive algorithm OPTIMUM in Algorithm 5.

The main observation is the sequence of functions is also split into two subproblems. Let $M_{a,b}$ be the optimum value on the row a. We consider the functions in two classes, $L = \{i|f_i(x_a) \leq f_i(x_b)\}$, and $R = \{i|f_i(x_a) > f_i(x_b)\}$. The algorithm would be correct if we pass down all functions. However, when we pass the function to the left recursion, during the evaluating values $M_{a',b'}$ where $a' \in [a_{min}, a]$ and $b' \in [b_{\min}, b]$, the contribution of the function f_i where $i \in R$ is apparent: $\min(f_i(x_{a'}), f_i(x_{b'})) = f_i(x_{b'})$. Namely, $\sum_{i\in R} \min(f_i(x_{a'}), f_i(x_{b'})) = \sum_{i\in R} f_i(x_{b'})$. A similar result holds for right recursion. Therefore, there is no need to pass down all the functions; instead, we sum the functions and pass them down. This ensures the sequence of functions passed down is partitioned in two, each with one more function appended.

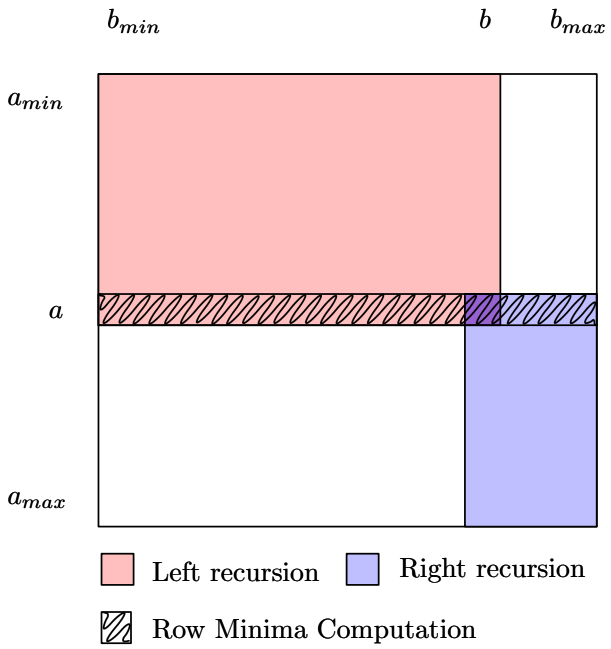

Figure 5.2 – A demonstration of one step of the recursion algorithm.

Algorithm 5 Algorithm for finding unimodal 2 medians. For each one of the procedures, the input f is a sequence of piecewise-linear unimodal functions.

function UNIMODAL2MEDIAN(f)
  Compute the global list of breakpoints $x_1, \ldots, x_n$
  Compute the † transform for each function
  return OPTIMUM($1, n, 1, n, f$)
end function

function OPTIMUM($a_{min}, a_{max}, b_{min}, b_{max}, f$)
  $a \leftarrow (a_{max} + a_{min})/2$
  $v, b \leftarrow$ ROWMINIMA(a, b_{min}, b_{max}, f)
  if $a_{min} = a_{max}$ **then**
    return v
  end if
  $L \leftarrow \{i | f_i(x_a) \leq f_i(x_b)\}$
  $R \leftarrow \{i | f_i(x_a) > f_i(x_b)\}$
  $f^L \leftarrow \{f_i | i \in L\} \cup (\sum_{i \in R} f_i)$
  $f^R \leftarrow \{f_i | i \in R\} \cup (\sum_{i \in L} f_i)$
  $v_L \leftarrow$ OPTIMUM($a_{min}, a, b_{min}, b, f^L$)
  $v_R \leftarrow$ OPTIMUM($a, a_{max}, b, b_{max}, f^R$)
  return $\min(v_L, v_R, v)$
end function

function ROWMINIMA(a, b_{min}, b_{max}, f)
  $f_1, \ldots, f_k$ are renumbered such that $f_j^\dagger(x_a) \leq f_{j+1}^\dagger(x_a)$
  $p \leftarrow 1$
  $A \leftarrow \{1, \ldots, k\}$
  for i from b_{min} to b_{max} **do**
    while $f_p^\dagger(x_a) < x_i$ **do**
      $A \leftarrow A \setminus \{p\}$
      $p \leftarrow p + 1$
    end while
    $v \leftarrow f_{\bar{A}}(x_a) + f_A(x_i)$
    if v is smallest seen value **then**
      $b \leftarrow i$
    end if
  end for
  return b
end function

Theorem 5.5.3. *Finding the optimum of Problem 2 for the input of k function of a total n breakpoints takes $O((n + k \log k) \log n)$ time.*

Proof. Define $T(n, m, k)$ to be the running time of the procedure OPTIMUM$(a_{min}, a_{max}, b_{min}, b_{max}, f)$ when $a_{max} - a_{min} + 1 = n$, $b_{max} - b_{min} + 1 = m$ and f is a sequence of k input functions.

Inside each recursive step, we spend $O(n+m+k \log k)$ time to find the row minima using Theorem 5.5.2. Additionally, we spend $O(m + n + k)$ time in total to partition the functions into left and right parts. Assume k_1 and k_2 are the number of functions in the left and right pieces of the recursion, then $k_1 + k_2 = k + 2$. Also, the number of breakpoints on the left and right is at most $m + 2$, as we duplicate at most 1 more breakpoints into the left and the right.

Hence, the running time satisfies the following recursive relation.

$$T(n, m, k) = \max_{\substack{m_1+m_2=m+2 \\ k_1+k_2=k+2}} T(n/2, m_1, k_1) + T(n/2, m_2, k_2) + O(m + n + k \log k).$$

Which solves to $T(n, m, k) = O((m + n + k \log k) \log n)$. Since $m = n$ in the top level of the recursion, we obtain a $O((n + k \log k) \log n)$ running time algorithm. □

Some additional optimization can be done that does not change the asymptotic worst-case running time but improves the running time in practice. For example, in the recursion, if at some point, the only function remaining is the function that came from a sum of original functions, then the recursion can stop earlier.

5.5.5 Data structure for piecewise-linear functions

We describe data structure over a set of piecewise-linear functions, and it can return the sum quickly. This data structure is required to implement ROWMINIMA, where two dynamic sums of piecewise-linear functions need to be maintained, and also OPTIMUM, where the sum of piecewise-linear function has to be computed and passed down.

Let $f_1, \ldots, f_k$ be piecewise-linear functions with a total of n distinct breakpoints. Although our algorithms can handle the case when breakpoints are not distinct, describing them does not provide additional insights.

Let $f_A = \sum_{i \in A} f_i$. Let a global set of points $x_1, \ldots, x_n$ contain all the breakpoints of each f_i. We want to maintain a data structure that maintains a set A and an index a, allowing fast evaluation of $f_A(x_a)$.

Formally, the data structure should have the following operations.

1. INITIALIZE($f_1, \ldots, f_k$): Process the functions $f_1, \ldots, f_k$, and return a data structure for $f_\emptyset$ and $a = -1$.
2. ADD(i): Update A into $A \cup \{i\}$.
3. REMOVE(i): Update A into $A \setminus \{i\}$.
4. EVALUATE(): Return $f_A(x_a)$.
5. NEXT(): Update a to $a + 1$.

Such data structure is standard, but we sketch it here for completeness.

Theorem 5.5.4. *Assuming all the breakpoints have been sorted, the data structure takes $O(n)$ time to construct, and any sequence of $O(n)$ queries takes $O(n)$ time.*

Proof. Initialization takes $O(n)$ time by merging the breakpoints of the functions.

Let $s_A(x)$ be the right slope of f_A at position x. The data structure will maintain $f_A(x_a)$ and $s_A(x_a)$ at all times.

EVALUATE() takes $O(1)$ time, since it just output $f_A(x_a)$ for the current value a.

ADD(i). There is no change in a but in A. Let's say we inserted i into A, and then we need to update the current information for $f_A(x_a)$, which is precisely adding $f_i(x_a)$ and updating $s_A(x_a)$ by adding $s_i(x_a)$. It is similar to deletion. Hence can be done in $O(1)$ time.

NEXT(). No change in A, but a change in a. The new $f_A(x_{a+1}) = f_A(x_a) + (x_{a+1} - x_a)s_A(x_a)$. The slope of $s_A(x_{a+1})$ either stays the same or changes if it is the breakpoint of a function indexed by A.

Since we process the breakpoints in order, there is no need to binary search, as we can always maintain a pointer to the largest breakpoint of f_i no larger than x_a for each i, so the total running time is $O(n)$. □

Theorem 5.5.5. *Problem 2 can be solved in $O((n + k \log k) \log n)$ time.*

Theorem 5.5.6. *The median split problem on k categories and n data points can be solved in $O((n + k \log k) \log n)$ time.*

5.6 Experiments

Our result is a theoretical exploration on the asymptotic complexity of solving the median split problem. However, we still experimented to see how it work in practice. We implemented the algorithm in C++ and ran it on an 8 Core AMD Ryzen 7 5800H with 16GB of ram. The input of each instance of the experiments are k sets, each contains 100 uniformly random integers, therefore $n = 100k$. n ranges through 10^6 to 3×10^7. We recorded the running time in the number of milliseconds. The result can be seen in Figure 5.3. Surprisingly, the time it takes looks linear. This can be partially explained by having optimizations that stop the recursion execution earlier.

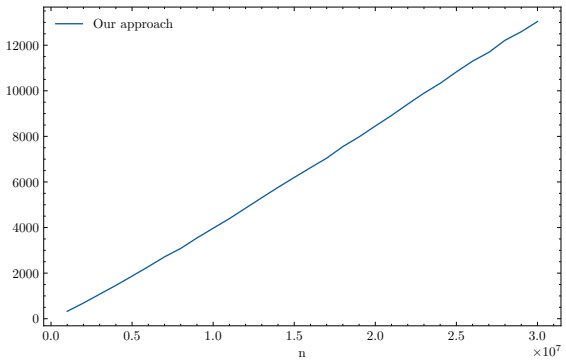

Figure 5.3 – Running time in ms vs number of data points.

Chapter 6

Conclusions and Perspectives

In this chapter, we conclude our exploration of improved and explainable decision tree-based models by reflecting on our key contributions, their implications, and the potential future directions for extending this work. In this work we have addressed complex challenges in model interpretability, particularly in handling categorical features with both convex and non-convex splitting criteria. Beyond summarizing our findings, this chapter offers reflections on the impact of our work, considerations of what might have been done differently, and thoughts on the path forward.

6.1 Summary of Contributions

We have introduced three key innovations that advance the field of decision tree learning and model interpretability:

— **Linear TreeSHAP**: A novel algorithm that computes Shapley values with linear time complexity, significantly improving the feasibility of interpreting large decision tree models, as compared to pre-existing work.

— **BSplitZ**: An innovative approach to integrating categorical features into decision tree models, utilizing the convex Zonotope framework to enhance both performance and

interpretability, especially in multiclass classification tasks.

— **Binary Splitting with MAE**: A novel approach to handling the lack of optimal numerical encodings for categorical features, addressing a longstanding challenge in decision tree research by providing a new perspective on splitting criteria.

6.2 Reflections on Real-World Applicability and Impact

While this thesis primarily focuses on theoretical advancements and algorithmic developments, the potential real-world applications of our contributions are noteworthy. Linear TreeSHAP, for instance, holds promise for diverse domains where interpretability is crucial. Although direct application to fields like healthcare and marketing was beyond the scope of this research, the algorithm's efficiency and scalability make it a strong candidate for such applications. Providing transparent feature attributions could support informed decision-making in diagnosing medical conditions or understanding customer preferences.

BSplitZ, with its capacity to handle complex categorical data in multiclass classification tasks, offers a versatile tool for practical applications in domains such as customer segmentation, text classification, and beyond. Its ability to improve decision tree splitting speed and accuracy makes it a valuable addition to the machine learning toolkit.

The work on binary splitting with MAE introduces a new perspective on handling categorical data in decision trees. While primarily theoretical, it lays the groundwork for further exploration into how categorical features can be effectively integrated into machine learning models, potentially influencing future research and applications.

Beyond the production of new methodologies for decision tree learning and interpretation, in this manuscript we have attained new perspectives, that highlight several key potential lessons and important considerations for the research community, notably:

— **Balancing Complexity and Interpretability**: One of the central challenges was achieving a balance between computational efficiency and model interpretability. The development of Linear TreeSHAP, for instance, required careful consideration

to ensure that the algorithm remained both practical for real-world applications and capable of providing meaningful insights.

— **Handling Categorical Data**: The process of developing BSplitZ and the binary splitting with MAE algorithm underscored the complexities inherent in working with categorical features in decision trees. This experience emphasized the importance of exploring innovative frameworks like the convex Zonotope to address these challenges effectively.

6.3 Alternatives, Future Directions and Perspectives

While the research has achieved its primary objectives, there are areas where different approaches could have been explored:

— **Extended Empirical Validation**: Conducting more extensive empirical validation across various domains could have provided a more robust demonstration of the practical applicability of the proposed algorithms. Deploying the algorithms in real-world settings would have offered deeper insights into their performance and impact.

— **Exploring Alternative Frameworks**: In developing BSplitZ, investigating alternative mathematical frameworks beyond the convex Zonotope could have yielded additional insights into handling categorical data. This might have uncovered other efficient methods for feature integration.

— **Incorporating User Feedback**: Engaging with end-users and stakeholders during the development process could have provided valuable feedback, especially for algorithms like Linear TreeSHAP, where interpretability plays a pivotal role. Such interaction might have influenced the design to better align with user needs and preferences.

Building on the foundations laid in this thesis, several avenues for future research are evident:

— **Expanding Linear TreeSHAP**: Future work could explore extending Linear Tree-

SHAP to other complex models, such as neural networks, to enhance interpretability across a wider range of machine learning models.

— **Advanced Handling of High-Cardinality Features**: Further research is needed to enhance the handling of high-cardinality categorical features within the BSplitZ framework. Incorporating statistical bound techniques could automate hyperparameter tuning, potentially improving performance.

— **MAE in Ensemble Methods**: Extending the application of MAE-based binary splitting to ensemble methods like gradient boosting machines could provide insights into its impact on model performance and interpretability in more complex architectures.

— **Ethical AI**: Investigating the ethical implications of model interpretability, particularly in ensuring fairness and mitigating bias in decision tree-based models, remains a crucial area for future exploration. Emphasizing how interpretability can contribute to more equitable AI systems is especially relevant.

6.4 Final Word

In conclusion, this work set out to deepen understanding of decision tree models and further, uncover insights into the complexities of machine learning and the importance of interpretability and ethical considerations in AI, and thus contribute to the ongoing dialogue in the field of explainable machine learning. Concretely, this manuscript offered novel solutions and perspectives, which give way to new paths of further exploration and innovation. These paths are filled with opportunities to enhance our understanding and application of machine learning models.

Bibliography

[1] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

[2] M. A. Ali, P. Hickman, and A. Clementson. The application of automatic interaction detection (aid) in operational research. *Journal of the Operational Research Society*, 26(2):243–252, 1975.

[3] T. C. Au. Random forests, decision trees, and categorical predictors: the" absent levels" problem. *Journal of Machine Learning Research*, 19(45):1–30, 2018.

[4] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete applied mathematics*, 65(1-3):21–46, 1996.

[5] D. Bertsimas and J. Dunn. Optimal classification trees. *Machine Learning*, 106:1039–1082, 2017.

[6] R. Blanquero, E. Carrizosa, C. Molero-Río, and D. R. Morales. On sparse optimal regression trees. *European Journal of Operational Research*, 299(3):1045–1054, 2022.

[7] R. N. Bracewell and R. N. Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-hill New York, 1986.

[8] L. Breiman. Some properties of splitting criteria. *Machine Learning*, 24(1):41–47, 1996.

[9] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[10] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration (preliminary version). In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 49–56, 1997.

[11] D. V. Carvalho, E. M. Pereira, and J. S. Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[12] J. Catlett. Mega induction: a test flight. In L. A. Birnbaum and G. C. Collins, editors, *Machine Learning Proceedings 1991*, pages 596–599. Morgan Kaufmann, San Francisco (CA), 1991.

[13] 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*, pages 785–794. ACM, 2016.

[14] M. Christoph. *Interpretable machine learning: A guide for making black box models explainable*. Leanpub, 2020.

[15] H. Cohen. *A course in computational algebraic number theory*, volume 8. Springer-Verlag Berlin, 1993.

[16] O. De Moor. Categories, relations and dynamic programming. *Mathematical Structures in Computer Science*, 4(1):33–69, 1994.

[17] F. Doshi-Velez and B. Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[18] C. Drummond and R. C. Holte. Severe class imbalance: Why better algorithms aren't the answer. In *European Conference on Machine Learning*, pages 539–546. Springer, 2005.

[19] J.-A. Ferrez, K. Fukuda, and T. M. Liebling. Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm. *European Journal of Operational Research*, 166(1):35–50, 2005.

[20] W. D. Fisher. On grouping for maximum homogeneity. *Journal of the American statistical Association*, 53(284):789–798, 1958.

[21] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528, 2003.

[22] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[23] C. Guo and F. Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

[24] H. Gupta and V. Asha. Impact of encoding of high cardinality categorical data to solve prediction problems. *Journal of Computational and Theoretical Nanoscience*, 17(9-10):4197–4201, 2020.

[25] K. Hamidieh. A data-driven statistical model for predicting the critical temperature of a superconductor. *Computational Materials Science*, 154:346–354, 2018.

[26] J. T. Hancock and T. M. Khoshgoftaar. Survey on categorical data for neural networks. *Journal of big data*, 7(1):28, 2020.

[27] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

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

[29] J. M. Hickey, P. G. Di Stefano, and V. Vasileiou. Fairness by explicability and adversarial shap learning. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*, pages 174–190. Springer, 2021.

[30] P. J. Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pages 492–518. Springer, 1992.

[31] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21, 1973.

[32] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[33] 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 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[34] L. Khachiyan, E. Boros, K. Borys, V. Gurvich, and K. Elbassioni. Generating all vertices of a polyhedron is hard. *Twentieth Anniversary Volume: Discrete & Computational Geometry*, pages 1–17, 2009.

[35] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Morgan Kaufman Publishing*, 1995.

[36] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 202–207. AAAI Press, 1996.

[37] Z. C. Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[38] W.-Y. Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

[39] W.-Y. Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348, 2014.

[40] W.-Y. Loh and Y.-S. Shih. Split selection methods for classification trees. *Statistica sinica*, pages 815–840, 1997.

[41] W.-Y. Loh and N. Vanichsetakul. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, 83(403):715–725, 1988.

[42] 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(1):56–67, 2020.

[43] S. M. Lundberg, G. G. Erion, and S.-I. Lee. Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*, 2018.

[44] M. Mampaey, S. Nijssen, A. Feelders, R. Konijn, and A. Knobbe. Efficient algorithms for finding optimal binary features in numeric and nominal labeled data. *Knowledge and Information Systems*, 42(2):465–492, 2015.

[45] W. McGinnis, hbghhy, W. Tao, andrethrill, C. Siu, C. Davison, and N. Bollweg. scikit-learn-contrib/categorical-encoding: Release for zenodo, Jan. 2018.

[46] A. Messalas, Y. Kanellopoulos, and C. Makris. Model-agnostic interpretability with shapley values. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–7, 2019.

[47] D. Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD explorations newsletter*, 3(1):27–32, 2001.

[48] R. Mitchell, E. Frank, and G. Holmes. Gputreeshap: Fast parallel tree interpretability, 2020.

[49] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[50] T. Ottmann, S. Schuierer, and S. Soundaralakshmi. Enumerating extreme points in higher dimensions. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 562–570. Springer, 1995.

[51] J. K. Park. *The Monge Array: An Abstraction and Its Applications*. PhD thesis, Massachusetts Institute of Technology, 1999.

[52] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pages 6638–6648, 2018.

[53] J. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[54] J. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[55] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1:81–106, 1986.

[56] 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*, pages 1135–1144, 2016.

[57] B. D. Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[58] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005.

[59] B. Roy. All about categorical variable encoding. *Towards Data Science*, 2019.

[60] C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

[61] S. Ruggieri. Efficient c4. 5 [classification algorithm]. *IEEE transactions on knowledge and data engineering*, 14(2):438–444, 2002.

[62] D. Ruppert. The elements of statistical learning: data mining, inference, and prediction, 2004.

[63] S. L. Salzberg. C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993, 1994.

[64] H. M. Sani, C. Lei, and D. Neagu. Computational complexity analysis of decision tree algorithms. In *Artificial Intelligence XXXV: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11–13, 2018, Proceedings 38*, pages 191–197. Springer, 2018.

[65] M. Scott, L. Su-In, et al. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30:4765–4774, 2017.

[66] C. Seger. An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing, 2018.

[67] N. B. Shank and H. Yang. Coupon collector problem for non-uniform coupons and random quotas. *the electronic journal of combinatorics*, pages P33–P33, 2013.

[68] L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[69] K. S. Sree, J. Karthik, C. Niharika, P. Srinivas, N. Ravinder, and C. Prasad. Optimized conversion of categorical and numerical features in machine learning models. In *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 294–299. IEEE, 2021.

[70] K. Stinson, D. F. Gleich, and P. G. Constantine. A randomized algorithm for enumerating zonotope vertices. *arXiv preprint arXiv:1602.06620*, 2016.

[71] E. Strumbelj and I. Kononenko. An efficient explanation of individual classifications using game theory. *The Journal of Machine Learning Research*, 11:1–18, 2010.

[72] E. Štrumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41:647–665, 2014.

[73] L. F. R. A. Torgo. *Inductive learning of tree-based regression models*. PhD thesis, Universidade do Porto. Reitoria, 1999.

[74] M. van Diepen and P. H. Franses. Evaluating chi-squared automatic interaction detection. *Information Systems*, 31(8):814–831, 2006.

[75] S. Verwer and Y. Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1625–1632, 2019.

[76] S. Weisbrich, G. Malissiovas, and F. Neitzel. On the fast approximation of point clouds using chebyshev polynomials. *Journal of Applied Geodesy*, 15(4):305–317, 2021.

[77] J. Yang. Fast treeshap: Accelerating shap value computation for trees. *arXiv preprint arXiv:2109.09847*, 2021.

[78] P. Yu, A. Bifet, J. Read, and C. Xu. Linear tree shap. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 25818–25828. Curran Associates, Inc., 2022.

[79] J. Zhang, Q. Sun, J. Liu, L. Xiong, J. Pei, and K. Ren. Efficient sampling approaches to shapley value approximation. *Proceedings of the ACM on Management of Data*, 1(1):1–24, 2023.

ECOLE DOCTORALE

Titre : Améliorer l'apprentissage des arbres décisionnels

Mots clés : Arbres de décision, Interprétabilité, Données catégorielles, Machine learning

Résumé : La modélisation par arbres de décision est reconnue pour son efficacité et sa lisibilité, notamment pour les données structurées. Cette thèse s'attaque à deux défis majeurs : l'interprétabilité des arbres profonds et la gestion des variables catégorielles. Nous présentons l'algorithme **Linear TreeShap**, qui facilite l'explication du processus décisionnel en attribuant des scores d'importance à chaque nœud et variable. Parallèlement, nous proposons un cadre méthodologique pour traiter directement les variables catégorielles, améliorant à la fois la précision et la robustesse du modèle. Notre approche inclut la méthode stochastique **BSplitZ**, conçue pour simplifier la répartition d'un grand nombre de catégories, et explore l'emploi du critère *Mean Absolute Error (MAE)*. Nous démontrons notamment l'inexistence d'un encodage optimal pour le MAE et résolvons un problème d'optimisation (le coût unimodal 2-median) essentiel aux opérations de scission. Ces travaux contribuent à la conception de modèles d'arbres de décision plus robustes et plus explicables, ouvrant de nouvelles perspectives pour l'apprentissage automatique.

Title : Improving Decision Tree Learning

Keywords : Decision trees, Interpretability, Categorical data, Machine learning

Abstract : Decision tree models are widely recognized for their efficiency and interpretability, particularly when working with structured data. This thesis addresses two main challenges: improving the interpretability of deep tree-based models and handling categorical variables. We introduce the **Linear TreeShap** algorithm, which illuminates the model's decision process by assigning importance scores to each node and feature. In parallel, we propose a methodological framework enabling decision trees to split directly on categorical variables, enhancing both accuracy and robustness. Our approach includes the stochastic **BSplitZ** method, designed to efficiently handle large sets of categories, and provides a thorough investigation of the *Mean Absolute Error (MAE)* criterion. In particular, we prove that no optimal numerical encoding exists under MAE and solve a related optimization problem (the unimodal cost 2-median) central to tree splitting. Our contributions advance the theoretical foundations and real-world applicability of decision tree models, paving the way for more robust and interpretable solutions in machine learning.

Institut Polytechnique de Paris
91120 Palaiseau, France