Pedro Sarmento Barbosa Martins

Decision Diagrams for Classification: New Constructive Approaches

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Thibaut Victor Gaston Vidal

Rio de Janeiro
March 2023

Pedro Sarmento Barbosa Martins

Decision Diagrams for Classification: New Constructive Approaches

Dissertation presented to the Programa de Pós–graduação em Informática da PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Rio de Janeiro, March 27th, 2023

Pedro Sarmento Barbosa Martins

Graduated in Economics from Universidade Federal do Rio de Janeiro, has worked in the software development field for over five years, and is currently a senior engineer and partner in a software startup in the financial market sector.

Bibliographic data

Sarmento Barbosa Martins, Pedro

Decision Diagrams for Classification: New Constructive Approaches / Pedro Sarmento Barbosa Martins; advisor: Thibaut Victor Gaston Vidal. – 2023.

64 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2023.

Inclui bibliografia

1. Informática – Teses. 2. Árvore de Decisão. 3. Diagrama de Decisão. 4. Aprendizado de Máquina. 5. Classificação. I. Vidal, Thibaut. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

To my family, for all the
support and encouragement.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Thibaut Vidal for his continuous support and guidance, both during the research itself and for his instrumental role in my admittance to the Master's program.

I would also like to extend my deepest gratitude to Prof. Alexandre Street (PUC-Rio), Prof. Marina Szapiro (UFRJ), and Prof. Manuel Gonzalo (UFRJ), who have guided me in different steps of my academic and professional journey, and to whom I owe a great deal.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Last but not least, I want to express profound appreciation to my family, for their instrumental encouragement and assistance in each and every step of the way.

Abstract

Sarmento Barbosa Martins, Pedro; Vidal, Thibaut (Advisor). **Decision Diagrams for Classification: New Constructive Approaches**. Rio de Janeiro, 2023. 64p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Decision diagrams are a generalization of decision trees. They have been repeatedly proposed as a supervised classification model for machine learning but have not been widely adopted. The reason appears to be the difficulty of training the model, as the requirement of deciding splits and merging nodes can lead to difficult combinatorial optimization problems. A decision diagram has marked advantages over decision trees because it better models disjoint binary concepts, avoiding the replication of subtrees and thus has less sample fragmentation in internal nodes. Because of this, devising an effective construction algorithm is important. In this context, the Optimal Decision Diagram (ODD) algorithm was recently proposed, which formulates the problem of building a diagram as a mixed-integer linear program (MILP), with a warm start provided by a greedy constructive heuristic. Initial experiments have shown that this heuristic can be improved upon, in order to find close-to-optimal solutions more effectively and in turn provide the MILP with a better warm start. In this study, we report improvements to this constructive heuristic, by randomizing the split decisions, pruning pure flows (i.e. flows with samples from a single class), and applying bottom-up pruning, which considers the complexity of the model in addition to its accuracy. All proposed improvements have positive effects on accuracy and generalization, as well as the objective value of the ODD algorithm. The bottom-up pruning strategy, in particular, has a substantial impact on the objective value, and thus on the ability of the MILP solver to find optimal solutions. In addition, we provide experiments on the expressiveness of decision diagrams when compared to trees in the context of small boolean functions in Disjoint Normal Form (DNF), as well as a web application for the visual exploration of the proposed constructive approaches.

Keywords

Decision Tree; Decision Diagram; Machine Learning; Classification.

Resumo

Sarmento Barbosa Martins, Pedro; Vidal, Thibaut. **Diagramas de Decisão para Classificação: Novas Abordagens Construtivas**. Rio de Janeiro, 2023. 64p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Diagramas de decisão são uma generalização de árvores de decisão, já propostos como um modelo de aprendizado de máquina para classificação supervisionada mas não largamente adotados. A razão é a dificuldade em treinar o modelo, já que o requerimento de decidir *splits* (partições) e *merges* (uniões de nós) em conjunto pode levar a problemas difíceis de otimização combinatória. Um diagrama de decisão tem importantes vantagens sobre árvores de decisão, pois melhor expressa conceitos binários disjuntos, evitando o problema de duplicação de subárvores e, portanto, apresentando menos fragmentação em nós internos. Por esse motivo, desenvolver algoritmos efetivos de construção é um esforço importante. Nesse contexto, o algoritmo *Optimal Decision Diagram* (ODD) foi recentemente proposto, formulando a construção do diagrama com programação inteira mista (MILP na sigla em inglês), com um *warm start* proveniente de uma heurística construtiva gulosa. Experimentos mostraram que essa heurística poderia ser aperfeiçoada, a fim de encontrar soluções próximas do ótimo de maneira mais efetiva, e por sua vez prover um *warm start* melhor. Nesse estudo, reportamos aperfeiçoamentos para essa heurística construtiva, sendo eles a randomização das decisões de *split*, a poda de fluxos puros (ou seja, fluxos de exemplos pertencentes a uma única classe), e aplicando uma poda *bottom-up* (de baixo para cima), que considera a complexidade do modelo além da sua acurácia. Todos os aperfeiçoamentos propostos têm efeitos positivos na acurácia e generalização, assim como no valor objetivo do algoritmo ODD. A poda *bottom-up*, em especial, tem impacto significativo no valor objetivo, e portanto na capacidade da formulação MILP de encontrar soluções ótimas. Ademais, provemos experimentos sobre a expressividade de diagramas de decisão em comparação a árvores no contexto de pequenas funções booleanas em Forma Normal Disjuntiva (DNF na sigla em inglês), assim como uma aplicação web para a exploração visual dos métodos construtivos propostos.

Palavras-chave

Árvore de Decisão; Diagrama de Decisão; Aprendizado de Máquina; Classificação.

Table of contents

List of figures

List of tables

List of algorithms

List of Abreviations

CART	Classification and Regression Trees
DNF	Disjunctive Normal Form
ODD	Optimal Decision Diagram
MILP	Mixed-Integer Programming

1
Introduction

Classical machine learning models have been used for several decades now. Their relevance has arguably increased in recent years, even while much attention has been given to advances in the deep learning landscape. Still, for tabular data, frequently used in fields as diverse as medicine, finance, and manufacturing, deep learning has not surpassed the performance of classical tree-based models such as Random Forests (SHWARTZ-ZIV; ARMON, 2022; GORISHNIY et al., 2021; GRINSZTAJN; OYALLON; VAROQUAUX, 2022). The growing application of algorithmic decision-making based on machine learning models in high-stake settings, including healthcare, credit-lending, and criminal justice, has also evidenced the need for simple, human-interpretable models (RUDIN, 2019; RUDIN et al., 2022).

Decision trees and related models are considered a cornerstone of classical machine learning for classification tasks. A decision tree can be interpreted as a sequence of recursive partitions of the training data feature space, where each partition — also called a *split* — can be visualized as the internal node of a tree. The tree leaves represent classes, and thus a path from the root of the tree to a leaf can be interpreted as a classification rule. Its ease of implementation, good performance on tabular data, and simplicity have made decision trees a popular choice for many applications (MOONEY, 2022).

High variance, a distinct shortcoming of decision trees (HASTIE; TIBSHIRANI; FRIEDMAN, 2001), has been commonly handled by the use of ensemble methods, which combine the output of many individual trees into a single model. The overfitting tendency of fully-grown trees has been eased by the use of pruning methods and stopping criteria. Nevertheless, a few problems inherent to the structure of a decision tree remain. In particular, trees are unable to express certain disjoint concepts, such as XOR, without growing entirely identical subtrees – and thus increasing the overall size of the tree. As a result, samples that statistically support split decisions are scattered across multiple nodes, weakening these supports. These are known as the replication and fragmentation problems, respectively (OLIVER, 1993).

Replication can be avoided if the tree structure is generalized to that of a *directed acyclic graph* (DAG). In DAGs, nodes are allowed to have *merges* besides splits, so that two different nodes may share a common child. In doing so, DAGs can also limit the number of nodes in the graph, as it does not grow exponentially with depth. A *decision diagram* is a classification model that generalizes decision trees by using a DAG topology. The model has thus some advantages over classic trees, as it avoids the subtree replication problem, does not grow exponentially with depth, and is less prone to fragmentation of data in internal nodes (OLIVER, 1993).

Several algorithms have been proposed for building decision diagrams (OLIVER, 1993; KOHAVI, 1994; KOHAVI; LI, 1995; OLIVEIRA; SANGIOVANNI-VINCENTELLI, 1996; PLATT; CRISTIANINI; SHAWE-TAYLOR, 2000; IGNATOV; IGNATOV, 2018). These algorithms are usually based on heuristics. Because of merges, decision diagrams are more difficult to implement and solve efficiently than decision trees. Early proposals would first generate a decision tree and then detect candidate nodes to merge, possibly leading to difficult combinatorial optimization problems, or resort to fixing the diagram topology a priori, restricting the learning procedure. As a result, decision diagrams have not been widely adopted over decision trees.

Recently, renewed attention has been given to algorithms for learning decision diagrams, especially using optimal methods instead of heuristics. Cabodi et al. (2021) and Hu, Huguet and Siala (2022), for instance, use *satisfiability solvers* (SAT) to build optimal binary decision diagrams. In a recent work co-authored by the present study author, the Optimal Decision Diagram (ODD) algorithm is proposed for learning decision diagrams with a *mixed-integer programming* (MILP) approach (FLORIO et al., 2022). The ODD training consists of two steps: a heuristic construction, which greedily builds an initial diagram using a top-down approach, and an optimization step which solves the complete MILP program using the previous step solution as a warm start.

The ODD algorithm constructs decision diagrams that are generally sparser than their decision tree counterparts and achieve better accuracy. However, experiments with data sets of practical importance were shown to take several minutes to complete before finding an optimal solution. Much of this computing time was spent in the second step of the algorithm. This indicates that the heuristic approach could be improved upon, in order to find close-to-optimal solutions more effectively, and in turn provide the MILP program with a better warm start. In addition to enhancing the optimal construction algorithm, an improved heuristic approach could also be used for larger problem instances (i.e., with thousands of features), which MILP struggles to solve efficiently.

In the present study, we analyze new constructive heuristics for decision diagrams, with special attention to the context of the ODD algorithm and the improvement of its heuristic step. The implemented improvements include randomization, pre-pruning, and post-pruning methods[1].

In addition to studying the proposed improvements, we also perform an experimental analysis of the expressiveness of decision diagrams when compared to decision trees. We train both models on uniformly sampled examples of random boolean functions expressed in *disjunctive normal form* (DNFs). The fact that any boolean function can be expressed as a DNF and that many concepts can be naturally represented by DNFs, such as medical diagnoses or tax rules, make them a suitable subject for expressiveness analysis.

Finally, we present a web application developed to visually debug the constructive approaches analyzed in this study. Besides visualizing the learned diagram with split and merge decisions for a particular data set, skeleton, and α hyperparameter, the tool allows constructive improvements to be individually activated or deactivated, log messages to be shown in a debug panel, and all constructive steps to be replayed.

The study is organized as follows. Chapter 2 discusses related works in the context of interpretability, decision trees, and decision diagrams. Chapter 3 defines the original constructive algorithm, its pitfalls, and the proposed improvements. It also discusses the expressiveness of decision diagrams in the context of DNFs. Chapter 4 provides the experimental results obtained for the proposed constructive approaches and the expressiveness comparison between decision diagrams and trees. Chapter 5 presents the visual explorer tool. Lastly, Chapter 6 concludes the study.

[1]Note that these improvements were not present in the first versions of the paper, but were incorporated by the time it was published in AAAI 2023.

2
Related Works

2.1
Interpretability: An Overarching Concern in Machine Learning

Machine learning research on classification has been traditionally focused on accuracy and other classifier performance metrics. Recently, the application of ML models in high-stakes decisions, such as healthcare, credit lending, and criminal justice, has brought questions of interpretability and explainability into the field (CARVALHO; PEREIRA; CARDOSO, 2019). There has been renewed interest in simple, logical models in the literature, such as decision trees and lists (RUDIN, 2019; RUDIN et al., 2022; CARRIZOSA; MOLERO-RÍO; MORALES, 2021).

The appeal of these models lies in the interpretability of their features. Humans can readily visualize and comprehend simple decision trees, while linear model weights may have a precise and easy-to-grasp interpretation. This inherent interpretability can be advantageous compared to black box models that demand separate models as explanations, as this type of black box explanation can be unreliable and misleading (RUDIN, 2019).

However, even defining interpretability and explainability in the context of machine learning has proven difficult, as these notions can be subjective and domain-specific, while the quality assessment of explanations is restricted by a lack of standard metrics and comparison frameworks in the machine learning literature (CARVALHO; PEREIRA; CARDOSO, 2019). In a given domain, interpretability and explainability needs may also differ across agents, including scientists, developers, stakeholders, and the final users of a model (TOMSETT et al., 2018). Finally, even models expected to be inherently interpretable can suffer from a lack of transparency. Large, complex decision trees can be less comprehensible than simple trees, while linear models with correlated features will have weights that are not easily interpretable.

Decision diagrams carry the same advantages for interpretability as decision trees — as well as their caveats. Their improved expressibility and generally sparser structure have potentially positive impacts on the model intelligibility when compared to trees. In this study, however, we will not investigate the interpretability of decision diagrams, computationally or empirically, but it is an important topic for future research.

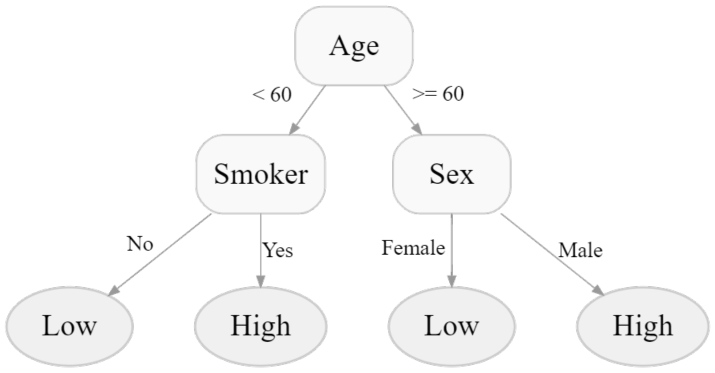


Figure 2.1: Example of a decision tree for a fictitious task of heart attack risk classification.

2.2 Decision Trees for Classification

Decision trees are a supervised learning model frequently used for classification and regression tasks. To define a decision tree, consider a set of classes $C = [1, 2, \ldots, k]$ and a data set X of learning examples (x, y), where each x is a vector $(x_1, \ldots, x_j)$ of feature values and $y \in C$ is the corresponding classification label. The root and each internal node in a learned decision tree represent a *split* in the feature space of X, and edges correspond to each resulting half-space. Leaf nodes represent classes in C. In that way, a new example x' can be classified by traversing the tree from its root and following the path dictated by the feature value of x' in each split, until reaching a leaf node. The class represented by this node is the learned class for x'.

An example of a decision tree for a fictitious data set can be seen in Figure 2.1. The task is to classify patients into high and low risk of heart attack, given their age, sex, and smoker status. If presented with a new example $x' = (75, \text{Male}, \text{Yes})$, it is trivial to follow the splits at each node, and reach the conclusion that x' should be classified as *high risk*.

Note that the decision tree can be interpreted as a set of rules for classification. In the example of Figure 2.1, such a rule for classifying a patient as *high risk* would be

$$(\text{Age} \geq 60 \ \wedge \ \text{Sex} = \text{Male}) \ \vee \ (\text{Age} < 60 \ \wedge \ \text{Smoker} = \text{Yes}). \qquad (2\text{-}1)$$

Decision trees have been widely adopted as a standalone classification model or as the base for *ensemble* models, such as Random Forests (BREIMAN, 2001) and Gradient Boosted Trees (FRIEDMAN, 2001). Since they can be easily visualized and also represented as classification rules, decision trees can offer a higher degree of transparency and interpretability when compared to more complex models, such as neural networks. Classical tree algorithms are easy to implement and have shown good performance on a variety of data sets, especially for tabular data.

2.2.1 Learning Decision Trees

Learning an optimal decision tree is known to be NP-hard (HYAFIL; RIVEST, 1976), but practical heuristic algorithms have proven effective in learning trees with good classification performance. These algorithms are usually top-down greedy procedures that make locally optimal split decisions at each node. The feature and threshold value for a split are selected based on an impurity measure, such as information gain or Gini impurity. The objective is to find the split that best separates the samples reaching a node, where a perfect split would separate the samples into subgroups consisting of a single class each. Stopping criteria and pruning methods are usually employed to limit the size and complexity of the learned tree.

Given the training data set X, the basic algorithm for learning decision trees will recursively partition its feature space, trying to keep samples belonging to the same class grouped together. At a node v, let X_v be the data that reaches it and $n_v = |X_v|$. Then, for each feature j in X, the samples are ordered in relation to j, and for each possible threshold value, t_v a candidate split $\theta = (j, t_v)$ is defined. The X_v data is partitioned into $X_v^{left}(\theta)$ and $X_v^{right}(\theta)$ subsets

$$X_v^{left}(\theta) = \{(x, y) \mid x_j \leq t_v\} \tag{2-2}$$

$$X_v^{right}(\theta) = X_v \setminus X_v^{left}(\theta). \tag{2-3}$$

The quality of all candidate splits at node v are then computed using a choice of impurity function H (such as entropy or Gini impurity)

$$G(X_v, \theta) = \frac{n_v^{left}}{n_v} H(X_v^{left}(\theta)) + \frac{n_v^{right}}{n_v} H(X_v^{right}(\theta)). \tag{2-4}$$

Next, the split that minimizes impurity is selected

$$\theta^* \in \operatorname{argmin}_\theta G(X_v, \theta). \tag{2-5}$$

Finally, the algorithm recurses for subsets $X_v^{left}(\theta)$ and $X_v^{right}(\theta)$. Stopping criteria, such as a maximum allowed depth or a minimum number of samples per node, are used to limit the recursion and the size of the tree.

Decision tree algorithms have a long history (see, for instance, LOH 2014). The ID3 (Iterative Dichotomiser 3) algorithm was developed in 1986 (QUINLAN, 1986), works solely for categorical features, and uses information gain as the impurity metric. The C4.5 algorithm (QUINLAN, 1993) is the successor of ID3. It extends support to continuous features by partitioning them into a discrete set of intervals and uses *gain ratio* as the measure of impurity. The C4.5 algorithm also applies post-pruning of the learned tree and deals with missing values.

The CART (Classification and Regression Trees) algorithm (BREIMAN et al., 1984) was developed in 1984 as an improvement to earlier decision tree induction methods such as AID (MORGAN; SONQUIST, 1963) and THAID (MESSENGER; MANDELL, 1972) and is still widely adopted. Instead of using stopping rules, it applies cost-complexity post-pruning (discussed in more depth in Section 3.3.1).

More recently, some attention has been given to the construction of *optimal* decision trees, pushed by advances in hardware and in combinatorial optimization software (BIXBY, 2012). The work of Bertsimas and Dunn (BERTSIMAS; DUNN, 2017) has been the root for continuous improvement of mathematical programming techniques for learning decision trees (CARRIZOSA; MOLERO-RÍO; MORALES, 2021), and other research paths include SAT (NARODYTSKA et al., 2018), dynamic programming (DEMIROVIĆ et al., 2020), and branch-and-bound search (AGLIN; NIJSSEN; SCHAUS, 2020). These techniques are usually limited to small and medium-sized problems, but the substantial improvements in combinatorial optimization software have yielded compelling results in data sets of practical importance.

2.2.2 Shortcomings

Without any technique for restraining size and complexity, decision trees tend to be large and overfit the training data, as the number of internal nodes in a tree grows exponentially with depth. Pruning methods and stopping criteria are thus usually employed to simplify decision trees, avoid overfitting, and improve generalization. Decision trees are also unstable, meaning that small variations in the data set X can lead to vastly different trees. This problem is usually mitigated by the use of ensemble methods, which combine the prediction of multiple decision trees.

Another disadvantage of decision trees is their inefficiency in expressing concepts such as XOR and thus solving higher-dimension boolean problems, such as parity, majority-on, and multiplexer. This can lead to *replication* and

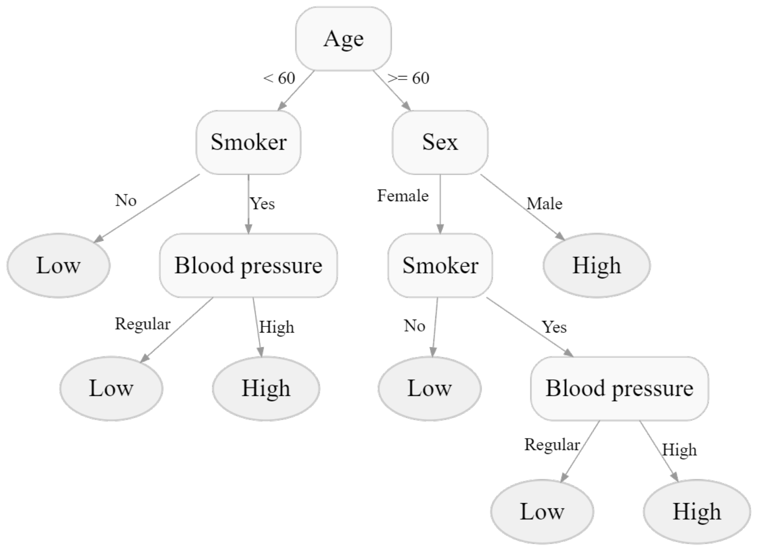


Figure 2.2: Example of a decision tree with subtree replication.

fragmentation issues in the learned tree (OLIVER, 1993; PAGALLO, 1989; VILALTA; BLIX; RENDELL, 1997).

To express such concepts as XOR, the learned tree is forced to replicate subtrees, which leads to a higher number of internal nodes. An example is given in Figure 2.2. This learned tree is a variation of the one found in Figure 2.1, with an added *Blood pressure* feature. The rule for classifying a sample as *high risk* has changed to

$$(\text{Age} \geq 60 \ \land \ \text{Sex} = \text{Male}) \ \lor \ (\text{Smoker} = \text{Yes} \ \land \ \text{Blood p.} = \text{High}). \quad (2\text{-}6)$$

Notice how the subtree that represents the second term in the proposition above has been duplicated. This happens because of the disjunctive form of the decision rule and the addition of a new feature.

Apart from creating more complex trees, another consequence of the replication problem is the partition of samples that support the replicated subtrees between different internal nodes, leading to weaker statistical support for each split decision. This is also known as the *fragmentation problem.*

Some proposals have been made to address the replication and fragmentation problems. These usually consist of learning an initial tree and then searching for possible feature combinations to create new features (PAGALLO, 1989; VILALTA; BLIX; RENDELL, 1997). A new tree learned on those features can then better express disjoint boolean concepts such as XOR. The approach has the disadvantage of needing to first build an initial tree and the possible loss of interpretability caused by using combined features.

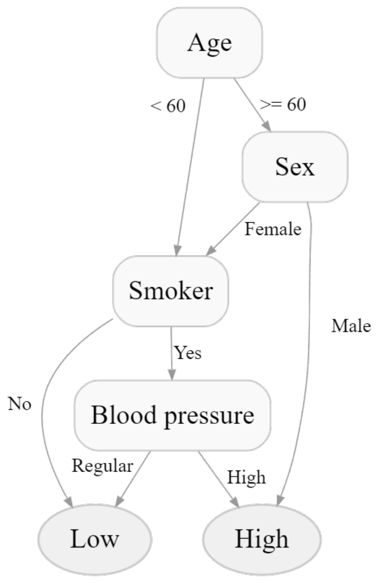


Figure 2.3: Example of a decision diagram.

2.3 Decision Diagrams

Decision diagrams are a generalization of decision trees. In addition to splits, diagrams also allow *merges*, meaning that two nodes can share a common child. This allows for multiple paths from the root to a leaf and has the advantage of not growing the number of nodes exponentially with depth. An example of a decision diagram is given in Figure 2.3. It represents the same classification rule of the tree in Figure 2.2.

The resulting diagram has fewer internal nodes than the corresponding tree in this case. Even though the set of decision functions representable by diagrams is the same as the set representable by trees (OLIVER, 1993), decision diagrams have a higher expressive power. This in turn avoids the replication and fragmentation problems found in decision trees.

Decision diagrams are also known as decision graphs or decision streams and have a history preceding and independent of their use as a classification model. They have been explored as a compact data structure for logical statements, being widely applied in symbolic logic and circuit validation (LEE, 1959; Bryant, 1986; BRYANT, 1992). Decision diagrams have also been applied in optimization (BEHLE, 2007; BERGMAN et al., 2016; LANGE; SWOBODA, 2021) and artificial intelligence in the context of planning (SANNER; UTHER; DELGADO, 2010; CASTRO et al., 2019), knowledge compilation (NIEUWENHUIS et al., 2012; LAI; LIU; WANG, 2013; SERRA, 2020), and constraint propagation (ANDERSEN et al., 2007; PEREZ; RÉGIN, 2015; VERHAEGHE; LECOUTRE; SCHAUS, 2018).

Decision diagrams have also been used in machine learning for tasks other than classification. The model has been applied to extend support vector machines for multiclass classification (PLATT; CRISTIANINI; SHAWE-TAYLOR, 2000) and as a surrogate model for neural networks (CHOROWSKI; ZURADA, 2011).

2.3.1 Decision Diagrams for Classification

Decision diagrams have not been widely adopted as a classification model, but several constructive approaches have been proposed. The inherent difficulty in learning diagrams arises from having to decide both splits and merges, and also from determining the best diagram topology. Therefore, most early algorithm proposals consisted of building a decision tree and then finding merges in a post-processing step (MAHONEY; MOONEY, 1991), or of fixing the diagram structure a priori (BAHL et al., 1989).

Mahoney and Mooney (1991) proposed to identify and merge related subtrees in a decision tree constructed using a standard tree learning algorithm. Their results showed limited generalization performance, and the task of finding related subtrees was found to be too computationally expensive. The work of Bahl et al. (1989) approached the problem by fixing the *gross structure* of the decision diagram a priori. As the topology of a decision diagram may be dependent on the specific domain and classification problem being solved, this a priori definition can be a considerable disadvantage.

Oliver (1993) tried to overcome both the related subtree and the fixed structure problems using a greedy bottom-up construction algorithm and the Minimum Message Length Principle (MMLP). At each iteration, each leaf in the graph defines a candidate split on a feature value, each leaf pair defines a candidate merge, and the MMLP is used to choose the best alteration (split or merge). If that alteration improves the diagram's message length, then the algorithm performs that alteration to the diagram. He reported better classification performance over decision trees on relatively simple problems, but further tests found the methodology to fail on more complex cases, where the algorithm would tend to perform premature joins (OLIVEIRA; SANGIOVANNI-VINCENTELLI, 1996).

Oliveira and Sangiovanni-Vincentelli (1996) also used the MMLP. Their approach builds *reduced ordered* decision diagrams, which are diagrams that have no redundant nodes and where the tests performed on the variables follow a fixed order for all paths in the graph. This allows the use of efficient algorithms known from logical synthesis for manipulating this kind of graph. From an initial reduced-order decision diagram, the algorithm derives a compact diagram by performing incremental changes until a local optimum is obtained.

Other approaches using reduced ordered decision diagrams have been proposed. In Kohavi (1994) the diagram is built in a bottom-up fashion, starting at the level closest to the terminal nodes. Kohavi and Li (1995) proposed an alternative algorithm that builds reduced ordered decision trees and then identifies and merges related subtrees.

Shotton et al. (2013) proposes ensembles of decision diagrams, called *decision jungles*. Split and merging decisions are optimized jointly by minimizing the weighted entropy sum at the leaves, and the algorithm is implemented using local search heuristics. The authors reported improved generalization compared to random forests while using substantially less memory.

A recent trend in the literature has been the proposal of *optimal methods* for the construction of decision diagrams for classification, in step with the rise of optimal methods for decision trees as reviewed in Section 2.2.1. Cabodi et al. (2021) present a SAT-based model for computing a decision tree as the smallest reduced ordered binary decision diagram. Hu, Huguet and Siala (2022) proposed a similar approach, with the key differences of using a MaxSAT solver and limiting the resulting diagram depth.

Simultaneously, another research stream in the optimal methods space is the Optimal Decision Diagram (ODD) algorithm (FLORIO et al., 2022), proposed by the current study author and others. As this study is closely connected to the ODD algorithm context, it will be described in more depth in the next section.

2.3.2 The Optimal Decision Diagram Algorithm

The ODD algorithm is the first MILP approach to train decision diagrams for classification (FLORIO et al., 2022). The model jointly optimizes merge and split decisions, while using a limited number of binary variables for increased efficiency. Because of the mathematical programming formulation of the model and its objective function, the approach can be easily extended for fairness, parsimony, and stability notions.

An important first step of the ODD algorithm is the construction of an initial decision diagram by a greedy top-down heuristic. This is needed primarily as a warm-start, that is, to provide the MILP solver with a feasible decision diagram candidate, preferably close to the optimal solution, in order to decrease its search space and improve its computing time. Aside from its use as a warm-start, the heuristic approach can be applied to large problems where finding an optimal solution would be too computationally expensive.

In the initial implementation of the ODD algorithm, experiments with data sets of practical importance were shown to take several minutes to complete before finding an optimal solution. Much of this computing time was spent in the second step of the algorithm, indicating that the heuristic approach could be improved upon. The original constructive heuristic, its flaws, and proposed improvements are discussed in more depth in Chapter 3.

2.3.2.1 Formulation

The MILP formulation must account for the flow variables that represent the trajectory of samples, the design variables that define the topology of the diagram, including merges, and the split decisions. The decision diagram is represented by an acyclic graph $G = (V, E)$. The set of nodes V is the union of the set of internal nodes V^I and the set of leaves V^C. Nodes are represented by indices $V = \{0, \dots, |V^I| + |V^C| - 1\}$, so that node $0 \in V^I$ is the diagram root, and the remaining nodes are listed by increasing depth, from left to right. Let V_l^I be the set of nodes at depth l, and let $\delta^-(v)$ and $\delta^+(v)$ be the sets of predecessors and successors of each node $v \in V$. The decision diagram produced by ODD will be a subgraph of G.

Figure 2.4 displays a visualization of this formulation scheme, with three layers of internal nodes and two terminal nodes. The thick edges indicate a possible decision diagram and the black connectors illustrate flow conservation within the graph. The ODD algorithm also permits long arcs between the black connectors of layers V_0^I and V_1^I and the terminal nodes of V^C, but they are not displayed in Figure 2.4 for clarity.

The following description of the MILP formulation omits certain constraints for the sake of brevity. These are indicated in the text and we refer to Florio et al. (2022) for the complete formulation.

Sample flow. Each sample i and internal node $u \in V^I$ is associated to a pair of flow variables $w_{iu}^- \in [0, 1]$ and $w_{iu}^+ \in [0, 1]$. Variables $z_{iuv}^- \in [0, 1]$ (respectively $z_{iuv}^+ \in [0, 1]$) characterize the flow going from the negative and positive sides of u to other nodes v. The following constraints express flow conservation within

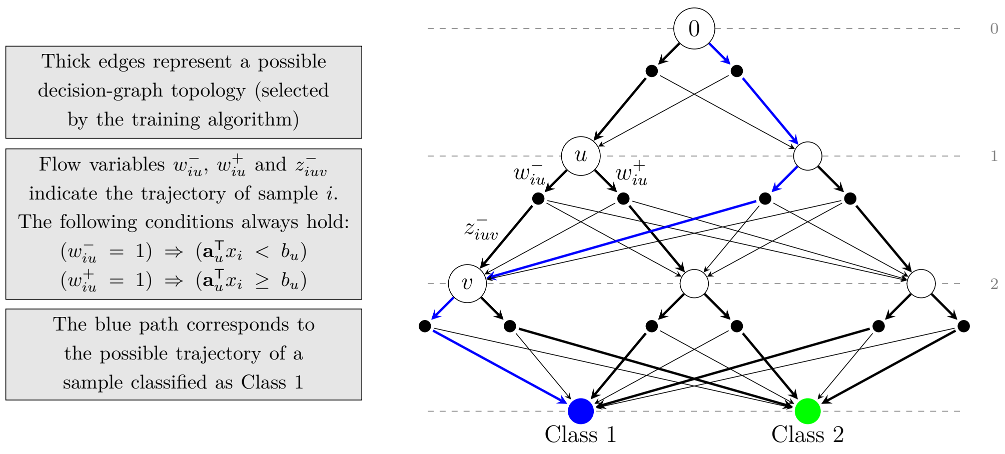

Thick edges represent a possible decision-graph topology (selected by the training algorithm)
Flow variables w^-_{iu}, w^+_{iu} and z^-_{iuv} indicate the trajectory of sample i. The following conditions always hold:
$(w^-_{iu} = 1) \Rightarrow (\mathbf{a}_u^\intercal x_i < b_u)$
$(w^+_{iu} = 1) \Rightarrow (\mathbf{a}_u^\intercal x_i \geq b_u)$
The blue path corresponds to the possible trajectory of a sample classified as Class 1
0
u
v
w^-_{iu}
w^+_{iu}
z^-_{iuv}
0
1
2
Class 1
Class 2

Figure 2.4: Visual example of the MILP formulation for the Optimal Decision Diagram algorithm.

the graph G, for each $u \in V^I$ and $i \in \{1, \ldots, n\}$:

$$w^+_{iv} + w^-_{iv} = \begin{cases} 1 & \text{if } v = 0 \\ \sum_{u \in \delta^-(v)} (z^+_{iuv} + z^-_{iuv}) & \text{otherwise} \end{cases} \tag{2-7}$$

$$w^-_{iu} = \sum_{v \in \delta^+(u)} z^-_{iuv} \tag{2-8}$$

$$w^+_{iu} = \sum_{v \in \delta^+(u)} z^+_{iuv}. \tag{2-9}$$

Additional constraints are needed to ensure integer sample flows (as it would not make sense to have *half* a sample going in or out of a node), but they are omitted here for the sake of brevity. Importantly, this formulation allows for fewer binary variables than a direct definition of the w^-_{iu} and w^+_{iu} as binary, improving the MILP efficiency.

Topology. Connecting the flow variables to the topology of the diagram, one binary variable $d_u \in \{0, 1\}$ is defined for each $u \in V$ that takes value 1 if this node is used in the classification (i.e., samples can pass through it). For the negative and positive sides of each node $u \in V^I$, binary design variables $y^-_{uv} \in \{0, 1\}$ and $y^+_{uv} \in \{0, 1\}$ take value 1 if and only if u links towards v on the negative and positive sides, respectively.

$$d_u = \sum_{v\in\delta^+(u)} y^+_{uv} = \sum_{v\in\delta^+(u)} y^-_{uv} \qquad u \in V^I \tag{2-10}$$

$$d_v \le \sum_{u\in\delta^-(v)} (y^+_{uv} + y^-_{uv}) \qquad v \in V^I - \{0\} \tag{2-11}$$

$$y^+_{uv} + y^-_{uv} \le d_v \qquad u \in V^I, v \in \delta^+(u) \tag{2-12}$$

$$z^+_{iuv} \le y^+_{uv}, \qquad u \in V^I, v \in \delta^+(u), i \in \{1,\dots,n\} \tag{2-13}$$

$$z^-_{iuv} \le y^-_{uv} \qquad u \in V^I, v \in \delta^+(u), i \in \{1,\dots,n\} \tag{2-14}$$

As it stands, the formulation allows for entirely symmetrical and equivalent topologies, a redundancy that significantly enlarges the MILP solver search space. Additional constraints for breaking symmetry are included in the ODD algorithm to improve efficiency but are omitted here for brevity.

Splits. Each internal node $v \in V^I$ is associated to a vector of variables $\mathbf{a}_v \in [-1,1]^d$ and a variable $b_v \in [-1,1]$ to characterize the splitting hyperplane. Samples $i \in \{1,\dots,n\}$ following the negative-side path should satisfy $\mathbf{a}_v^\intercal \mathbf{x}^i < b_v$, whereas samples taking the positive-side path should satisfy $\mathbf{a}_v^\intercal \mathbf{x}^i \ge b_v$. This is done by including *indicator constraints* that express the following implication logic for each $i \in \{1,\dots,n\}$ and $v \in V^I$:

$$(w^-_{iv} = 1) \Rightarrow (\mathbf{a}_v^\intercal \mathbf{x}^i + \varepsilon \le b_v) \tag{2-15}$$

$$(w^+_{iv} = 1) \Rightarrow (\mathbf{a}_v^\intercal \mathbf{x}^i \ge b_v), \tag{2-16}$$

where ε in Constraint (2-15) should be a small constant greater than the numerical precision of the solver.

Constraints (2-15–2-16) represent general multivariate splits. To produce univariate splits, a binary variable $e_{vj} \in \{0,1\}$ is added for each feature $j \in \{1,\dots,d\}$ and internal node $v \in V^I$. These constraints ensure the selection of a single feature:

$$\sum_{j=1}^{d} e_{vj} = 1 \qquad v \in V^I \tag{2-17}$$

$$-e_{vj} \le a_{jv} \le e_{vj} \qquad j \in \{1,\dots,d\}, v \in V^I \tag{2-18}$$

2.3.2.2 Objective Function

The ODD algorithm optimizes accuracy and an additional regularization term that penalizes complex decision diagrams with a large number of internal nodes. To compute the model accuracy, binary variables $w_{iv} \in [0,1]$ are defined for each sample $i \in \{1,\dots,n\}$ and leaf $v \in V^C$ expressing the amount of flow of i reaching terminal node v with class c_v. These variables must satisfy the following constraints for $v \in V^C$ and $i \in \{1,\dots,n\}$:

$$w_{iv} = \sum_{u \in \delta^-(v)} (z^+_{iuv} + z^-_{iuv}) \tag{2-19}$$

With these variables in place, the ODD algorithm objective can be defined as

$$\min \frac{1}{n} \sum_{i=1}^{n} \sum_{v \in V^C} \phi_{iv} w_{iv} + \frac{\alpha}{|V^I| - 1} \sum_{v \in V^I - \{0\}} d_v, \tag{2-20}$$

where ϕ_{iv} represents the mismatch penalty when assigning sample i to terminal node v (for a typical classification task, it may be defined as 0 if $c^i = c_v$ and 1 otherwise), while α is a regularization parameter. The first term of the objective penalizes misclassified samples, whereas the second term penalizes complex models.

3
Constructive Approaches

In this chapter, we define the basic heuristic construction algorithm, as well as its pitfalls, and the proposed improvements. The improved implementations are used in the computational experiments of Section 4.1.

We also discuss the expressiveness of boolean functions in decision diagrams, represented in *disjunctive normal form* (DNFs). A comparison between diagrams and trees in this context is done in Section 4.2.

3.1
Original Construction

The decision diagram is built with a top-down approach similar to CART (BREIMAN et al., 1984). Borrowing from the notation introduced in Section 2.3.2 for the ODD algorithm, the diagram is here represented by a directed acyclic graph $G = (V, E)$, with nodes $V = V^I \cup V^C$ being the union of internal nodes V^I and leaves V^C. Nodes are represented by increasing indices $\{0, \ldots, |V^I| + |V^C| - 1\}$, from top to bottom and left to right, starting at the root node $0 \in V^I$.

A fixed *skeleton* of the diagram must be provided as input, which defines the maximum number of internal layers and nodes per layer. Departing from CART and other tree induction methods, the connections between these layers are not fixed and must be defined during training. The first layer always has a single node, the root, while a terminal layer is automatically assigned to the diagram, with one leaf node for each class in the data set. The input data set is expected to be numerical[1], with samples $X \in \mathbb{R}^{n \times d}$ and targets $y \in C^n$, where C is a set of classes $\{1, \ldots, K\}$.

Skeletons can be represented in text as a sequence, where each value represents the number of nodes in the respective diagram layer. The skeleton 1-2-4-4-4, for instance, represents a diagram structure with five internal layers and 15 internal nodes (including the root), while the number of nodes in the terminal layer depends on the number of classes K in the intended data set. In this way, a skeleton defines a set of internal layer indices $L = \{0,\ \ldots,\ m\}$, as well as a maximum number of nodes per layer w_l, $\forall l \in L$. The set V^I_l contains all nodes belonging to layer l.

The constructive method is described in Algorithm 1. For each internal layer $l \in L$ and node $v \in V^I_l$ from top to bottom, we greedily select the

[1]It requires prior transformation for categorical data, such as one-hot encoding.

Algorithm 1: Initial decision diagram constructive heuristic

```
Input: Data set: $X \in \mathbb{R}^{n \times d}$, $y \in C^n$, $K$, Skeleton: $G$, $L$, $w_l$ and $V_l^I\ \forall l \in L$
Output: Decision diagram: SplitFeature($v$), SplitThreshold($v$),
        Edge($v$,side) for each node $v \in V^I$ and side (positive/negative)
1  $X(0) \leftarrow \{1, \dots, n\}$                /* all samples arrive at root node */
2  foreach layer $l \in L$ do
3      Arcs($l$) $\leftarrow \emptyset$
4      foreach node $v \in V_l^I$ do
5          find $i^*$, $j^*$ that minimizes:
             $H^w(X(v), i, j),\ \forall i \in \{1, \dots, n\}, j \in \{1, \dots, d\}$
6          SplitFeature($v$) $\leftarrow j^*$
7          SplitThreshold($v$) $\leftarrow X_{i^*j^*}(v)$
8          $a^+(v) \leftarrow$ positive branch, sample indices $\{k \mid X_{kj^*}(v) \geq X_{i^*j^*}(v)\}$
9          $a^-(v) \leftarrow$ negative branch, sample indices $\{k \mid X_{kj^*}(v) < X_{i^*j^*}(v)\}$
10         insert $a^+(v)$ and $a^-(v)$ into Arcs($l$)
11     while $|$Arcs($l$)$| > w_{l+1}$ do
12         $H^m(a_1, a_2) \leftarrow H(X_{a_1} \cup X_{a_2}) - [H(X_{a_1}) + H(X_{a_2})]$
13         find $(a_1^*, a_2^*)$ that minimizes $H^m(a_1, a_2),\ \forall (a_1, a_2) \in$ Arcs($l$)
14         Remove $a_1^*$ and $a_2^*$ from Arcs($l$)
15         $a^* \leftarrow$ arc with branches and samples from the union of $a_1^*$ and $a_2^*$
16         Insert $a^*$ into Arcs($l$)
17     $i \leftarrow 0$
18     foreach arc $\in$ Arcs($l$) do
19         if layer $l + 1$ is the terminal layer then
20             $u \leftarrow$ leaf representing the majority class of samples from arc
21         else
22             $u \leftarrow i^{th}$ node in layer $l + 1$
23             $i \leftarrow i + 1$
24         $X(u) \leftarrow$ samples from arc
25         foreach branch $\in$ arc branches do
26             Edge($v$,branch side) $\leftarrow$ branch node
```

univariate split that minimizes the weighted entropy,

$$H^w(X^s(v), i, j) = H(X^s_{k<i,j}(v)) + H(X^s_{k \geq i,j}(v)), \tag{3-1}$$

where i is the sample that defines the splitting threshold, j is the splitting feature, and $X^s(v)$ is the set of samples that arrive at v, sorted by the value of feature j. The entropy H is defined as

$$H(X) = -|X| \sum_{c \in C} p(c) \log p(c), \tag{3-2}$$

where X is a set of samples, $c \in C$ is a target class, and $p(c)$ is the probability of class c, here defined as the frequency of occurrence of c in X.

From the optimal splitting sample i^* and splitting feature j^* of a node v, we can also define a pair of *arcs* departing v. An arc represents a flow of samples, where each sample may follow a single *branch* that departs from a node. A node has exactly two branches, termed *positive* and *negative*, as defined by the splitting hyperplane: samples k where $X^s_{kj^*}(v) \geq X^s_{i^*j^*}(v)$ follow the positive branch, while samples k where $X^s_{kj^*}(v) < X^s_{i^*j^*}(v)$ follow the negative branch. Arcs(l) is the set of all arcs departing layer l, and each arc can be represented by a tuple of two lists, one of node branches and another of samples.

After defining the splits and arcs for all nodes in a layer, we must decide how each arc connects to the subsequent layer. This is achieved with a greedy merging policy: as long as the number of arcs departing layer l is greater than the number of nodes in layer $l+1$, we merge the pair of arcs that least increases entropy, as defined by Equation 3-2. Merging a pair of arcs a_1 and a_2 consists of replacing them with a new arc a^*, whose node branches are the union of the node branches of a_1 and a_2, and whose samples are the union of the samples from both arcs.

We must then define the edges departing from each node of layer l. Because of the merging policy just described, the number of arcs departing layer l equals the number of nodes in layer $l+1$, so each arc $a \in$ Arcs(l) is assigned to a corresponding node $u \in V_{l+1}$ in order from left to right. An edge (v, u) is created for each node v in arc a, and node u receives all samples connected to this arc. In particular, if $l+1$ is the terminal layer of the diagram, each arc is directed to the node representing the majority class of its samples.

In this way, the algorithm is designed to minimize information impurity measured by entropy on each split and merge decision, while redirecting sample flows in the last layer to its respective predominant class, in order to find a solution with low misclassification.

Figure 3.1 displays examples of diagrams built using this constructive approach, as well as improved versions using the pruning methods described in Section 3.3. The diagrams displayed have a 1-2-4-4-4 skeleton and were constructed for the *tic-tac-toe*, *credit-approval*, and *parkinsons* data sets (see Section 4.1 for a description of the data sets), using the original algorithm, and modified by the pure flow pruning strategy and the bottom-up pruning strategy with $\alpha = 0.1$. The labels for the internal nodes are their respective indices[2]. The color saturation represents the proportion of samples that passes through that node, and the root is by definition reached by 100% of the samples. The metrics displayed for each diagram are respectively the training accuracy, the

[2]The rendering was automated using GraphViz (<https://graphviz.org/>), which defines the rendering order of the nodes.

test accuracy, and the objective value.

3.2 Pitfalls

The main issue with the original construction algorithm is the complexity of the resulting models. Given a skeleton, the heuristic always constructs a full diagram – i.e., one using all available nodes in the given skeleton –, regardless of whether it is advantageous to do so or not. Besides this, in the context of the ODD algorithm, the top-down constructive approach lacks a fundamental connection to the optimization problem solved in the MILP phase, since it does not consider the regularization parameter α. In consequence, one way to improve the heuristic is to simplify the diagram by pruning nodes that do not contribute to the ODD objective value or to the overall classification performance. In this context, we propose two pruning methods in Section 3.3.

The greedy aspect of the algorithm also undermines its performance. It always chooses the best local decision without look-ahead or hindsight, which may lead to suboptimal global solutions. Randomization methods are common strategies for this problem, and our proposed approach in Section 3.3 attempts to increase solution variability by randomizing the algorithm decisions.

Another issue, particular to decision diagrams and the original constructive heuristic, is the occurrence of *double arcs*. As is noticeable in Figure 3.1 (b-2), some internal nodes connect both their left and right arcs to the same child node. This does not constitute a proper split decision, since the feature space is not partitioned in that step of the model. Moreover, this type of flow is prohibited by the MILP formulation of the ODD algorithm. The issue can occur when greedy merging decisions during top-down construction select such a merge of both arcs of a node as being the best one. The pruning methods proposed in this study, particularly bottom-up pruning, effectively remove most double arcs in our experiments. As a future research path, dedicated strategies could be added to guarantee their removal.

3.3 Improvements to the Initial Heuristic

3.3.1 Pruning

Pruning is a classical method for simplifying decision trees. It does so by getting rid of subtrees that are superfluous for the model performance. Pruning methods have been analyzed extensively in the literature and have been shown

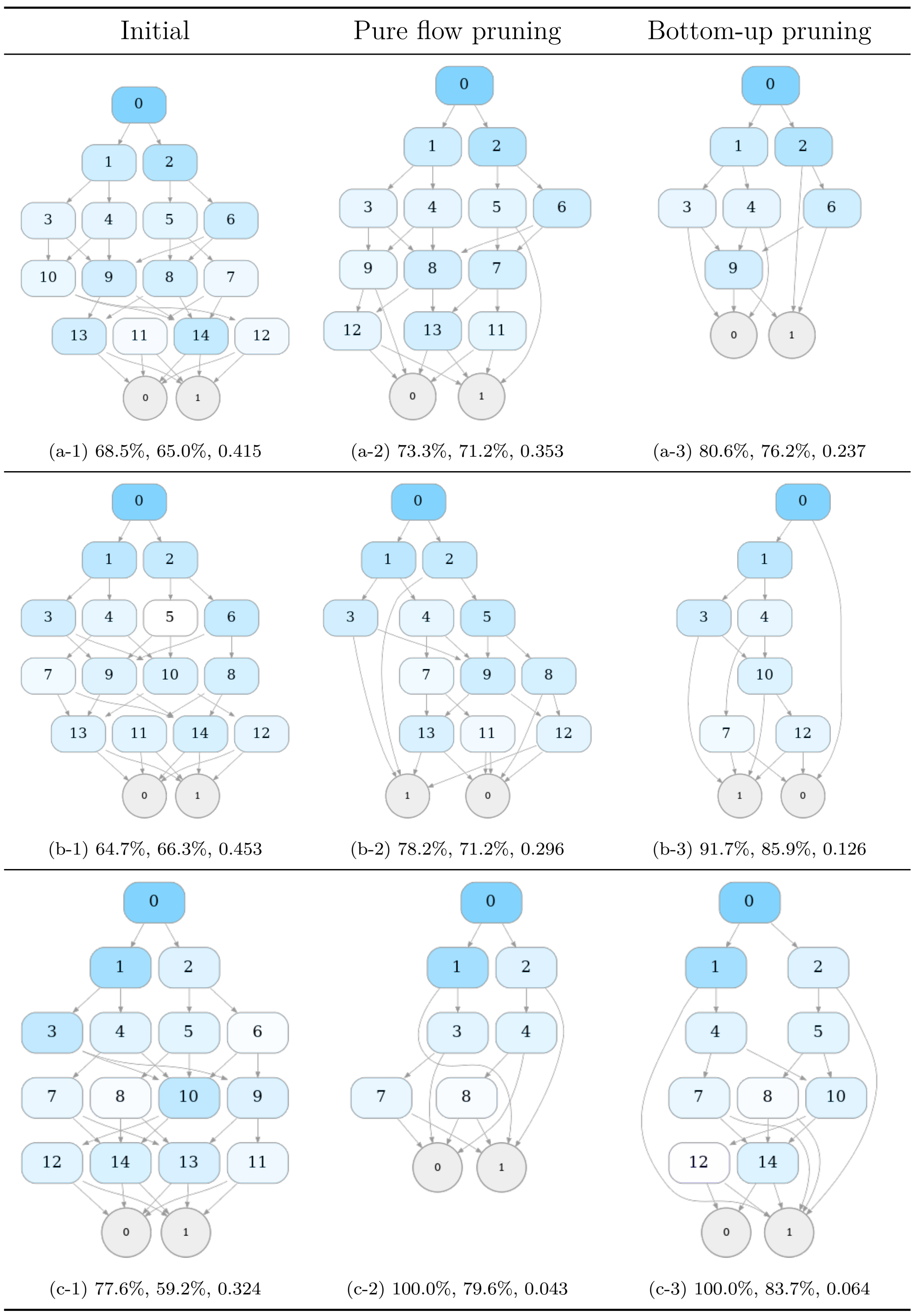

Initial
Pure flow pruning
Bottom-up pruning
(a-1) 68.5%, 65.0%, 0.415
(a-2) 73.3%, 71.2%, 0.353
(a-3) 80.6%, 76.2%, 0.237
(b-1) 64.7%, 66.3%, 0.453
(b-2) 78.2%, 71.2%, 0.296
(b-3) 91.7%, 85.9%, 0.126
(c-1) 77.6%, 59.2%, 0.324
(c-2) 100.0%, 79.6%, 0.043
(c-3) 100.0%, 83.7%, 0.064

Figure 3.1: Decision diagrams with a 1-2-4-4-4 skeleton constructed for the (a) *tic-tac-toe*, (b) *credit-approval*, and (c) *parkinsons* data sets by the heuristic algorithm and each pruning strategy. The metrics displayed for each diagram are respectively the training accuracy, the test accuracy, and the objective value.

to improve accuracy, especially in domains with noise (MINGERS, 1989; BRESLOW; AHA, 1997; COSTA; PEDREIRA, 2022). More importantly, pruning can improve the generalization of decision trees. A full-grown tree tends to overfit the training data (i.e., model its noise), and thus might perform poorly on out-of-sample data. Pruning fixes overfitting by limiting the model variance. Additionally, by simplifying a decision tree, pruning may improve its interpretability.

Pruning methods can be generally classified as *pre-pruning* or *post-pruning*. Pre-pruning is the application of early stopping procedures during the construction phase, such as a maximum depth or a minimum number of samples per node. Another classic procedure is to impose a minimum threshold on the test selection measure (QUINLAN, 1986; BRESLOW; AHA, 1997). For instance, Quinlan (1986) uses the Chi-squared test for stochastic independence as a stopping criterion in the ID3 algorithm. Pre-pruning is known to suffer from the horizon effect (BREIMAN et al., 1984), that is, the stopping criterion might terminate the constructive phase in a step, even if the same criterion would not prohibit subsequent steps. For this reason, post-pruning methods have been more readily adopted (BRESLOW; AHA, 1997).

Post-pruning, sometimes simply called pruning, involves growing a decision tree to its full extent and then eliminating superfluous subtrees according to some criteria. Reduced Error Pruning (REP), introduced by Quinlan (1987) is one of the simplest post-pruning methods. For each non-leaf subtree, the subtree is replaced by the best possible leaf and a new misclassification error is calculated over a separate validation set. If the resulting tree has a lower or equal error, the replacement is made to the original tree. This process continues until any further pruning increases the misclassification error. Despite its simplicity, REP is rarely used in practice because of the need for a separate validation set and its tendency to overprune trees (ELOMAA; KAARIAINEN, 2011).

Another classic strategy is known as *minimal cost-complexity pruning*, applied originally in the CART algorithm (BREIMAN et al., 1984). Given a fully constructed tree T_0 and a subtree T of T_0, we define the misclassification cost of the subtree as $R(T)$ and the set of leaf nodes of T as $\tilde{T}$. Then its cost-complexity measure $R_\alpha(T)$ is given by

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|, \tag{3-3}$$

where $\alpha \in \mathbb{R}$ is a cost penalty for complexity (here characterized as the number of leaf nodes in T). The minimal cost-complexity algorithm first defines a subtree T_1 by trivially removing leaves of T_0 whose misclassification is the same as that of their parent node. Then, it continues by recursively finding a sequence of subtrees $T_1 > T_2 > T_3 > \cdots > T_k$ that decreases in size, with respectively increasing parameters $0 = \alpha_1 < \alpha_2 < \alpha_3 < \cdots < \alpha_k$[3]. Finally, the algorithm selects one of these subtrees as the optimum-sized tree, as given by an estimate of the true misclassification cost.

In this study, we apply two pruning strategies to improve the original heuristic. The first is a pre-pruning method that takes advantage of the fixed leaves in the decision diagram structure. The second is a cost-complexity post-pruning that uses a bottom-up strategy. These strategies are discussed in more depth in the next sections.

3.3.1.1 Pure Flow Pruning

Consider a split that exclusively groups samples of the same class on one of its branches. In a topology with fixed leaf nodes corresponding to a class, all these samples can be simply directed to the leaf representing the respective class. This is a form of pre-pruning we have termed *pure flow pruning*. Notice it is not a stopping criterion as the constructive procedure continues normally for the other split branch and the other splits on the layer.

Besides *perfectly* pure sample flows, this method can be calibrated to permit softer thresholds. A branch with 90% or 95% of samples corresponding to a single class could be pruned in the same way, allowing for a simpler diagram at the cost of some accuracy. In this study, we focus solely on strictly pure flow pruning.

In Algorithm 1, implementing this pre-pruning method consists of extending the condition at Line 19. Even if layer $l + 1$ is not the terminal layer, we check if at least a certain ratio γ of the samples in the given arc is from a single class c, and, if so, define the receiving node u as the leaf representing c. In this study, as mentioned, we only use $\gamma = 1.0$ when evaluating this pruning strategy.

The second column of Figure 3.1 displays examples of decision diagrams constructed with this improvement strategy. In the topmost example, for instance, internal node 5 had a flow directed to node 7 that consisted solely of samples from a single class. In the improved solution, this flow is directly routed to the leaf node for this class, simplifying the diagram. In addition, both training and test accuracy improved, as well as the objective value.

[3]We refer to the original work of Breiman et al. (1984) for the details of how these trees and α parameters are constructed.

3.3.1.2 Bottom-up Pruning

Notice that the Optimal Decision Diagram (ODD) algorithm (described in Section 2.3.2) optimizes the diagram for both accuracy and complexity. The former is modeled in terms of the misclassification at the leaves and the latter is represented by the number of internal nodes used, being controlled by a hyperparameter α. When α is 0, only accuracy is considered. As α increases, solutions with a large number of nodes are penalized and simpler diagrams are encouraged.

We propose a bottom-up pruning method that closely follows the ODD objective and is outlined in Algorithm 2. This strategy removes nodes that connect to leaves when the gain in the regularization term of the objective value is higher than the loss of accuracy, given a choice of α. It is tailored-made for improving the warm-start solution for the optimization step of the ODD algorithm, since optimizing the objective value directly decreases its upper bound, restricting the search space of the mixed-integer programming solver.

The approach is close to the cost-complexity pruning method described in Section 3.3.1, by considering both accuracy and node count, even though the implementation details differ significantly.

Algorithm 2: Bottom-up pruning

```
Input: Hyperparameter $\alpha$, initial solution $G$, number of layers $m$,
       $V_l^I\ \forall l \in L$
Output: Updated decision diagram with pruned nodes
1  PruningGain ← $\frac{\alpha}{|V^I|}$
2  foreach layer $l \in \{m-1, \dots, 1\}$ do
3      foreach node $v \in V_l^I$ that connects to a leaf node do
4          Accuracy ← current solution accuracy
5          AccuracyIfPruned($v$) ← accuracy if node $v$ is pruned
6          AccuracyCost ← Accuracy − AccuracyIfPruned
7          if PruningGain ≥ AccuracyCost then
8              remove node $v$ from $G$
9              foreach parent node $w$ of $v$ do
10                 $u$ ← leaf node for the majority class of the samples from
                   $w$
11                 remove edge $(w, v)$ from $G$
12                 add edge $(w, u)$ to $G$
```

The last column of Figure 3.1 provides examples of this improvement strategy. All double arc issues are solved with this method, and the decision diagram is generally sparser. Training accuracy, test accuracy, and objective value are also improved in these solutions, compared to the initial implementation. Additionally, the fragmentation of samples through the diagram is lower than in the other solutions.

3.3.2 Randomization

Purely greedy constructive heuristics always make the best decision on each iteration. Besides being straightforward to implement and test, these deterministic algorithms are expected to reach reasonably good solutions once the construction is finished. Locally optimal decisions, however, disregard the global structure of the problem and are likely to reach only locally optimal solutions.

Randomization techniques can be used to improve a constructive heuristic by exploring promising but suboptimal decisions on each iteration, according to some randomized strategy. For problems of a reasonable size, randomized heuristics can be run several times in order to select the best overall solution. This procedure allows a larger portion of the search space to be explored, while still avoiding the exponential cost of an exhaustive search.

Randomization has been frequently used in decision tree construction methods, particularly in the context of tree ensemble models. These models fit many individual trees and combine them by weighted or unweighted voting, frequently resulting in better classification performance than individual trees. The main objective of tree ensemble methods is to decrease the variance of individual decision tree. This reduction in variance is only possible if the ensembled trees are diverse, which is achieved by randomization.

Common candidates for randomization in decision trees are the set of training samples and the split decisions. A frequently applied method for sample randomization is *bagging* (BREIMAN, 1996; DIETTERICH, 2000b), short for *bootstrap aggregating*. Bootstrap is the generation of multiple training sets by uniformly sampling from the original set with replacement. By sampling with replacement, each generated set is independent of the others, reducing variance. In bagging, the bootstrapped sets are used to fit individual models, which are then aggregated by voting to achieve a final classification.

Several methods have been proposed to randomize split decisions. Ali and Pazzani (1996) proposed a stochastic hill-climbing procedure that selects one split among a set of splits close to the optimal up to a margin β, with probability proportional to its information gain. Tin Kam Ho (1998) selects a random subspace of the feature space by keeping a small number of its dimensions, while fixing all feature values to a constant in the unselected dimensions, and proceeds to fully train a decision tree using the training samples projected to this subspace. The decisions of several individual trees are then combined by averaging the conditional probability of each class at the leaves. Dietterich (2000a) proposes a modified the C4.5 algorithm by choosing uniformly randomly among the 20 best splits with a non-negative information gain ratio. Cutler and Zhao (2001) and Geurts, Ernst and Wehenkel (2006) go one step further and propose methods where the split decision is made entirely at random, without optimizing over the splitting feature or its value. This ensures that the individual trees are largely independent, reducing the variance of the final, aggregated model significantly.

One of the most successful tree-based ensemble models are Random Forests (BREIMAN, 2001). The algorithm uses traditional bagging as well as *feature bagging*, where split decisions are made on random samples of features instead of the entire feature space. These steps greatly reduce the correlation between estimators in the final ensemble.

For decision diagrams, much like decision trees, the split decision of each internal node is a good candidate for randomization. In this study, we achieve this by feature bagging, as in Random Forests, restricting the number of features to be considered for each split decision. For this purpose, we repeatedly select a uniformly sampled ratio $r \in [0.5, 1.0]$ and solve the heuristic by considering a random subset of $r \cdot d$ features for each split decision, where d is the total number of features. This procedure is repeated for 10 seconds and the solution with the best objective value is selected. Notice that, contrary to Random Forests, our goal is to keep only a single decision diagram.

3.4 Expressiveness of Decision Diagrams

Decision trees are known to poorly model boolean disjoint concepts, such as XOR, causing the replication and fragmentation problems described in Chapter 2. Decision diagrams can better express these concepts, so an improved classification performance is expected for data sets where they occur.

Any n-bit boolean function $f\colon \{0,1\}^n \to \{0,1\}$ can be expressed in *disjunctive normal form* (DNF). In a space of d boolean variables $\{x_1, \cdots, x_d\}$, a DNF is a disjunction $T_1 \vee T_2 \vee \cdots \vee T_s$ of conjunctions of boolean literals $T_1 \ldots T_s$, which are made up of one or more of the d variables and their negations. An s-term DNF is one which has at most s conjunctions (also known as terms). A k-DNF is one in which each term is of size at most k, where $k \leq d$. The following is an example of an s-terms k-DNF function in a space of 10 variables (d), with 3 terms (r) and 4 variables (k) per term:

$$(x_3 \wedge x_6 \wedge \neg x_7 \wedge x_{10}) \vee (\neg x_1 \wedge x_5 \wedge x_7 \wedge \neg x_9) \vee (x_2 \wedge \neg x_4 \wedge \neg x_8 \wedge x_{10}) \quad (3\text{-}4)$$

As illustrated in Section 2.2.1, decision trees can be interpreted as classification rules, by describing the set of decisions from the root to each leaf. In binary classification tasks, this representation is commonly referred to as *decision rule sets*, which are essentially DNFs (FÜRNKRANZ; GAMBERGER; LAVRA, 2012). Each conjunction of the DNF can be considered as an individual rule, the rules are unordered, and the positive class is predicted if at least one of the rules is satisfied.

Rule sets have been found to be more interpretable to humans in general when compared to alternative representations, such as decision lists (rules ordered in an IF-ELSE sequence) or decision trees (LAKKARAJU; BACH; LESKOVEC, 2016), which aligns with the informal notion that DNFs are a natural representation of human knowledge (QUINLAN, 1993; VIKTOR; CLOETE, 1995).

In addition, DNF-type concepts – i.e., those that can be represented by small DNFs – have been shown to cause the replication problem in decision trees. Pagallo (1989) and Vilalta, Blix and Rendell (1997), for instance, have studied DNF learning in the context of decision trees and the replication problem, respectively proposing feature construction and global data analysis as solutions. Decision diagrams can be considered an alternative approach for this problem space.

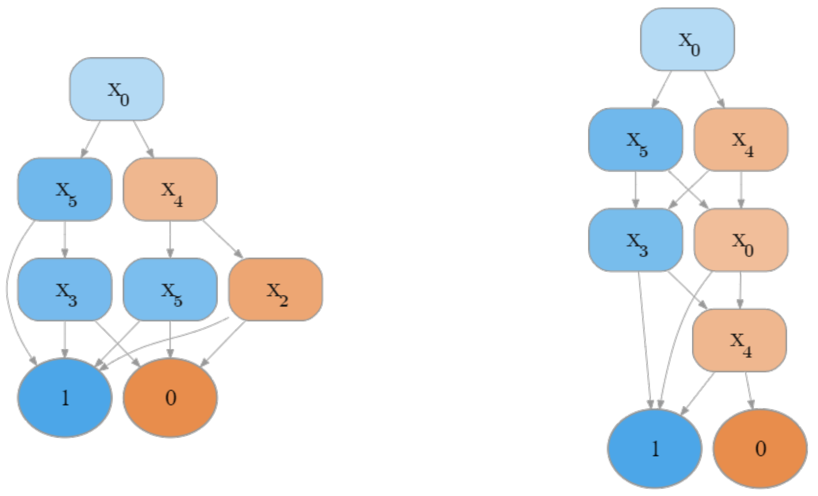
Because of its expressiveness, DNF learning has been extensively studied in the context of complexity and learning theory. In his seminal work on PAC-learning, Valiant discusses the learnability of restricted and unrestricted DNFs (VALIANT, 1984). In a subsequent work, Pitt and Valiant (1988) proved that properly learning an s-term DNF is NP-hard for $s \geq 2$.

Nonetheless, some positive results exist for restricted variations of DNFs. In terms of the distributional setting, quasi-polynomial time algorithms are known for learning DNFs under the uniform distribution (VERBEURGT, 1990; BLUM et al., 1994; JACKSON, 1997). For monotone DNFs (i.e., DNFs with no negated literals), random poly(n)-size formulas can be learned in probabilistic polynomial time (SELLIE, 2008; JACKSON et al., 2008). Furthermore, Sellie (2009) extended exact learning for general poly(n)-size DNFs in probabilistic polynomial time.

Figures 3.2a and 3.2b respectively show a decision tree and a decision diagram constructed using the algorithm in Section 3.1 and pure flow pruning.

The tree has skeleton 1-2-4 and the diagram 1-2-2-2. Both models were trained on uniformly random samples generated for the following random 6-term 2-DNF over 7 variables:

$$(x_4 \wedge x_6) \vee (x_4 \wedge x_5) \vee (x_3 \wedge x_4) \vee (\neg x_2 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_0 \wedge \neg x_4) \quad (3\text{-}5)$$



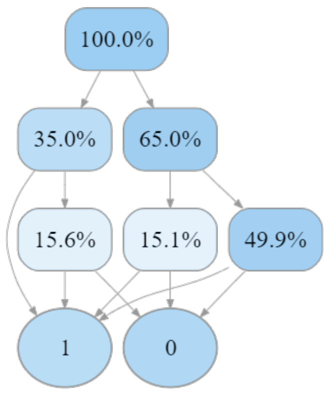
(a) Decision tree
Training accuracy: 75%
Test accuracy: 75%

(b) Decision diagram
Training accuracy: 90%
Test accuracy: 90%

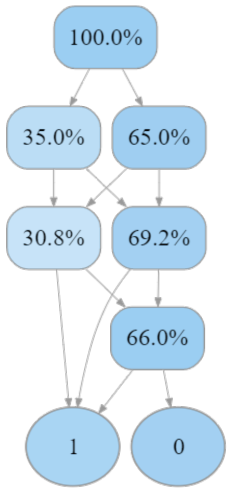
Figure 3.2: Models built with pure flow pre-pruning, trained on uniformly random samples for a random 6-term 2-DNF over 7 variables. The left branch of a split indicates a logical 1, the right branch indicates a logical 0.

The decision tree does not replicate subtrees in this example, mainly because of the pre-pruning of nodes with flows from a single class. Both models have the same number of internal nodes, but the decision diagram achieves significantly better classification performance. This appears to be generally true for small DNFs, as demonstrated by the experiments on Section 4.2.

Figure 3.3 compares the internal node fragmentation of the same learned models presented in Figure 3.2. The displayed values simply indicate the percentage of samples that flow through that internal node. Notice how the sample flow for a path through the decision tree necessarily decreases in each layer. Because of merges, this is not true for the sample flow through a decision diagram. In the fragmentation analysis of Section 4.2, we call this percentage of samples the *support* of a node. As indicated by the analysis results, decision diagrams produce models with significantly less overall fragmentation (higher average support per node) than decision trees.



(a) Decision tree



(b) Decision diagram

Figure 3.3: Internal node fragmentation in models built with pure flow pre-pruning, trained on uniformly random samples for a 6-term 2-DNF over 7 variables. The value displayed in an internal node is the percentage of training samples that flow through that node.

4
Experimental Analyses

In this chapter, we present our experimental analyses. The first analysis is related to the heuristic improvements discussed in Section 4.1. The second is a study on the classification performance and fragmentation of decision diagrams on small s-terms k-DNFs, in comparison to decision trees.

4.1
Constructive Heuristic Improvements

In this section, we provide the experimental setup and the results for the heuristic improvement experiments.

4.1.1
Experimental Setup

The computational experiments were conducted on 54 data sets from the UCI Machine Learning Repository (DUA; GRAFF, 2017), which are described in Table 4.1. These data sets were also used in the ODD algorithm study (FLORIO et al., 2022) and have an interesting diversity in the number of samples (n), features (d), and classes (c). A set of four random seeds $\{1, 2, 3, 4\}$ was selected, which controls the splitting of the data sets into training, validation, and test sets in the respective proportion of 50%, 25%, and 25%. The random seed also controls the split randomization logic for runs where this improvement is activated. Two base skeletons were chosen, a tree-like (1-2-4-8) and a diagram-like (1-2-4-4-4), both with 15 internal nodes. We consider five *heuristic versions*: one for the initial heuristic, one for each improvement discussed in Section 3.3 (bottom-up pruning, pure flow pruning, and randomized splits), and one for the complete heuristic with all improvements.

We also chose a set of α hyperparameters, $\{0.0, 0.1, 0.5, 0.9, 1.0\}$. For comparisons of objective value, we simply run each possible combination of the described configurations, and average the resulting objective values over them. For comparisons of accuracy, the computational experiments have two stages. First, we calibrate the regularization parameter α for each combination of data set, heuristic version, skeleton, and seed, using the validation set. Lastly, we use the best-found α hyperparameter to calculate training and test accuracies. In total, there were 10,800 different experiment runs.

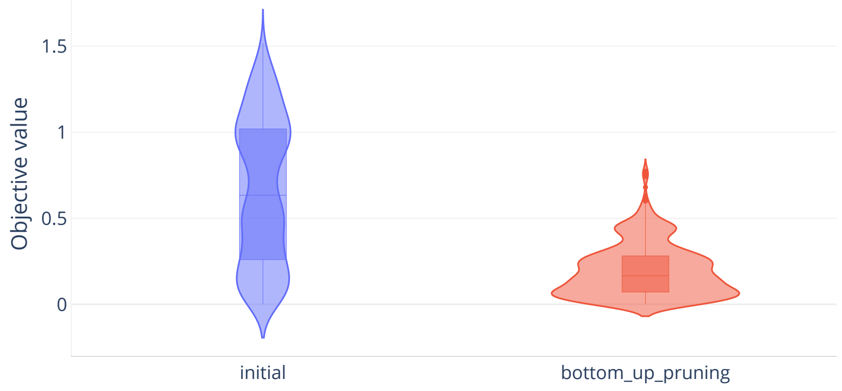
Table 4.1: The 54 datasets used in the computational experiments for the heuristic improvements.

Dataset	n	d	c	Dataset	n	d	c
acute-inflam-nephritis	120	6	2	iris	150	4	3
acute-inflam-urinary	120	6	2	mammographic-mass	830	10	2
balance-scale	625	4	3	monks1	556	11	2
banknote-auth	1372	4	2	monks2	601	11	2
blood-transfusion	748	4	2	monks3	554	11	2
breast-cancer-diag	569	30	2	optical-recognition	5620	64	10
breast-cancer-prog	194	33	2	ozone-eighthr	1847	72	2
breast-cancer-wisconsin	699	9	2	ozone-onehr	1848	72	2
car-evaluation	1728	15	4	parkinsons	195	22	2
chess-kr-vs-kp	3196	37	2	pima-indians-diab	768	8	2
climate-simul-crashes	540	18	2	planning-relax	182	12	2
congressional-voting	435	16	2	qsar-biodegradation	1055	41	2
connect-mines-rocks	208	60	2	seeds	210	7	3
connect-vowel	990	10	11	seismic-bumps	2584	20	2
contraceptive-method	1473	11	3	soybean-small	47	35	4
credit-approval	653	37	2	spambase	4601	57	2
cylinder-bands	277	484	2	spect-heart	267	22	2
dermatology	366	34	6	spectf-heart	267	44	2
echocardiogram	61	9	2	statlog-german-credit	1000	48	2
fertility-diagnosis	100	12	2	statlog-landsat-sat	6435	36	6
habermans-survival	306	3	2	teaching-assist-eval	151	52	3
hayes-roth	160	4	3	thoracic-surgery	470	24	2
heart-disease-cleveland	297	18	5	thyroid-ann	3772	21	3
hepatitis	80	19	2	thyroid-new	215	5	3
image-segmentation	2310	18	7	tic-tac-toe	958	18	2
indian-liver-patient	579	10	2	wall-following-robot-2	5456	2	4
ionosphere	351	33	2	wine	178	13	3

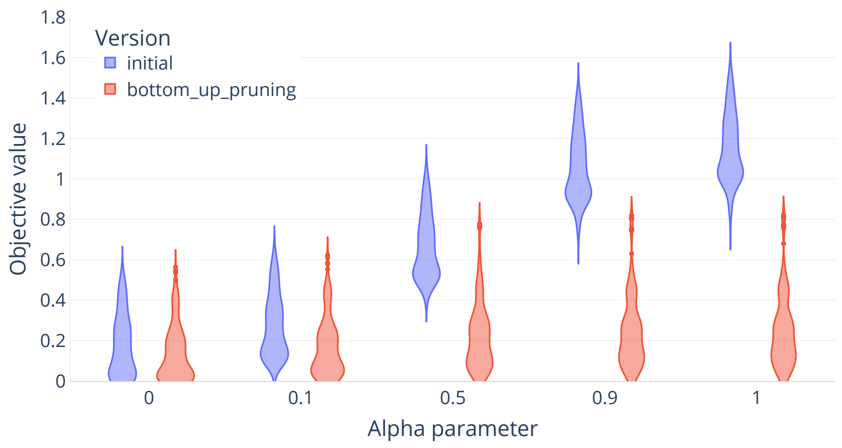
4.1.2 Bottom-Up Pruning

The post-pruning improvement to the heuristic directly optimizes the decision diagram for accuracy (low misclassification) and model complexity (proportional to the number of internal nodes and dependent on the α hyperparameter). It is expected to have a substantial impact on the objective value as calculated for the ODD algorithm in Section 2.3.2.2.

Figure 4.1 presents the results of applying bottom-up pruning for the objective value metric. The graphs in Figure 4.1a are overlaid *violin* and *box* plots. The box plot represents summary statistics for the underlying data, where the bottom of the box is the 25th percentile, the top is the 75th percentile, the line inside the box is the median, and the lines drawn from the bottom and the top of the box respectively reach the minimum and the maximum data points. The violin plots depict the *probability density* of the data at different values and help visualize its distribution.



(a) All experiment configurations.

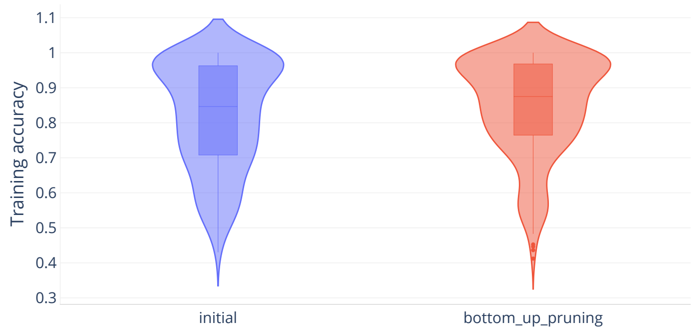


(b) Grouped by alpha parameter level.

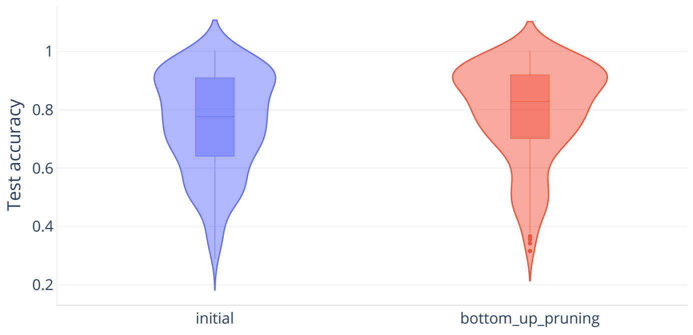
Figure 4.1: Impact of bottom-up pruning on objective value.

In Figure 4.1a, the overall effect of the post-pruning approach for all experiment configurations clearly demonstrates the improvement in the objective value. The effect is statistically significant under a paired Wilcoxon signed-rank test (p-value < 0.001). Figure 4.1b further confirms the effect by comparing the results of the initial and improved implementation for each α parameter value. When optimizing for simpler diagrams (α close to 1), bottom-up pruning finds significantly closer-to-optimal solutions.

The effects of bottom-up pruning on accuracy and generalization are less noticeable but still present. As Figure 4.2a indicates, many experiment configurations have yielded higher accuracy when post-pruning is applied. Pruning is also known to counter overfitting, so generalization is expected to improve. Figure 4.2b shows that this is true for most of the experiment configurations. A pairwise Wilcoxon signed-rank test indicates that these effects are also statistically significant (p-values < 0.001 for both metrics).



(a) Training accuracy.



(b) Test accuracy.

Figure 4.2: Impact of bottom-up pruning on training and test accuracy.

4.1.3 Pure Flow Pruning

Pure flow pre-pruning simplifies diagrams by directing sample flows of a single class to its corresponding leaf. The resulting simpler model retains its accuracy while improving its objective value as calculated by the ODD algorithm in Section 2.3.2.2.

The effects of pure flow pruning on the objective value are presented in Figure 4.3. While less substantial than the one for bottom-up pruning, this effect is noticeable, with denser distribution towards lower values.

Pure flow pre-pruning has a generally positive impact on accuracy and generalization. The impact on generalization is less significant than the one from bottom-up post-pruning, which is expected for a pre-pruning method. The effects of the constructive improvement on training and test accuracy are indicated in Figure 4.4.

The described effects of pure flow pruning on objective value, training accuracy, and test accuracy have been shown to be statistically significant by pairwise Wilcoxon signed-rank tests (p-values < 0.001).

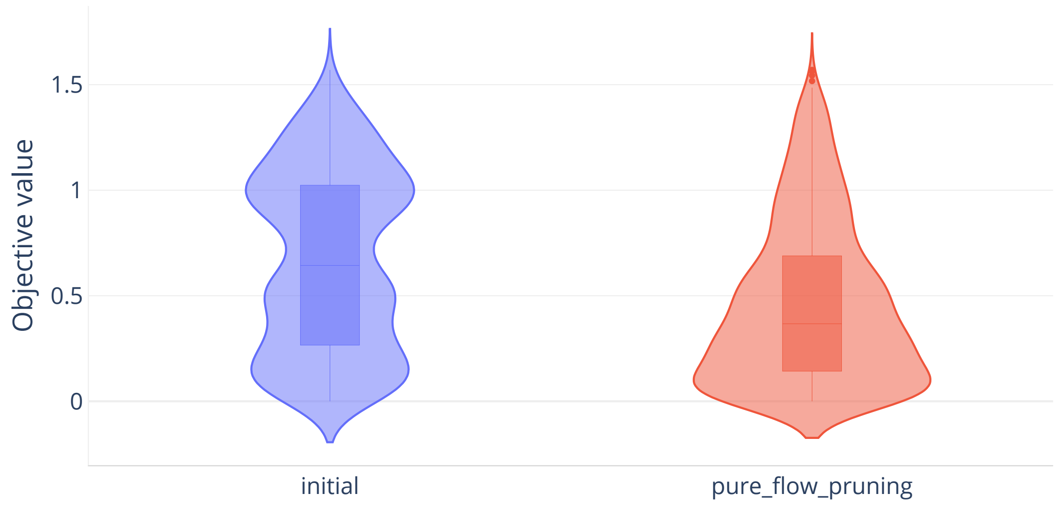
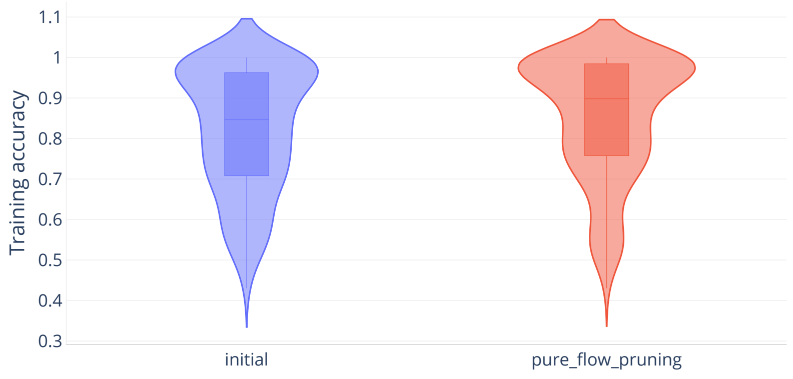
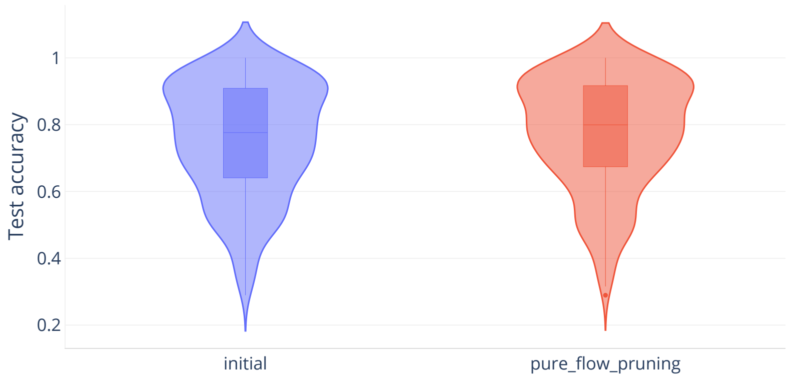


Figure 4.3: Impact of pure flow pruning on objective value.



(a) Training accuracy.

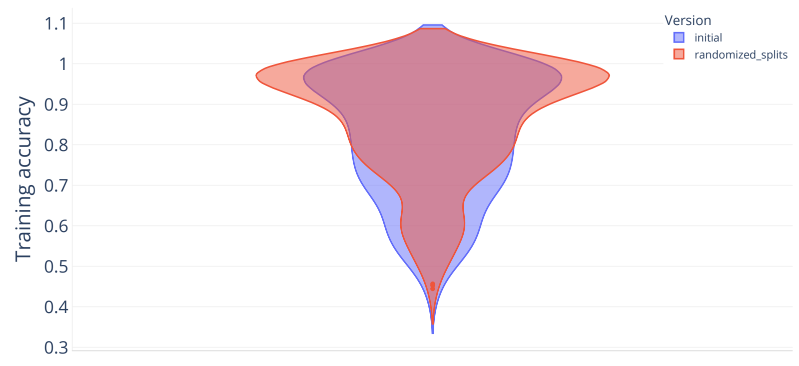


(b) Test accuracy.

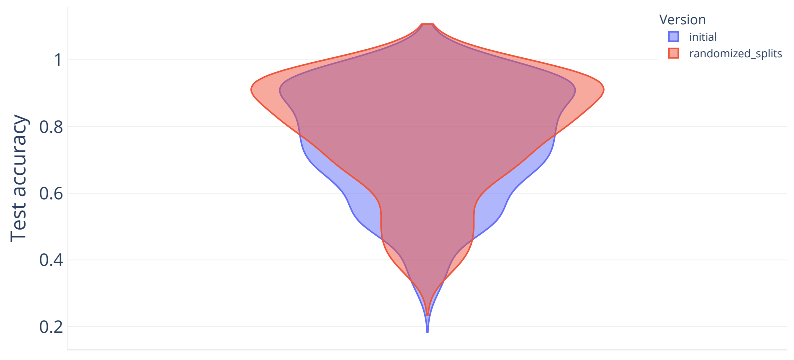
Figure 4.4: Impact of pure flow pruning on training and test accuracy.

4.1.4 Randomized Splits

Randomization allows for the constructive algorithm to optimize over a broader solution-space, instead of relying solely on local optimal decisions, without the penalty of an exhaustive search. In decision diagrams, randomizing splits should improve accuracy and generalization. This is precisely the effect evidenced in Figure 4.5.



(a) Training accuracy.



(b) Test accuracy.

Figure 4.5: Impact of randomizing splits on training and test accuracy.

For the objective value, randomizing splits does not yield substantial results. Figure 4.6 presents the distribution for this experimental case. There appears to be a slight improvement, with more solutions found with lower objective values, although the overall distribution remains the same.

Pairwise Wilcoxon signed-rank tests indicate that the effects of randomized splits on objective value, training accuracy, and test accuracy are statistically significant (p-values < 0.001).

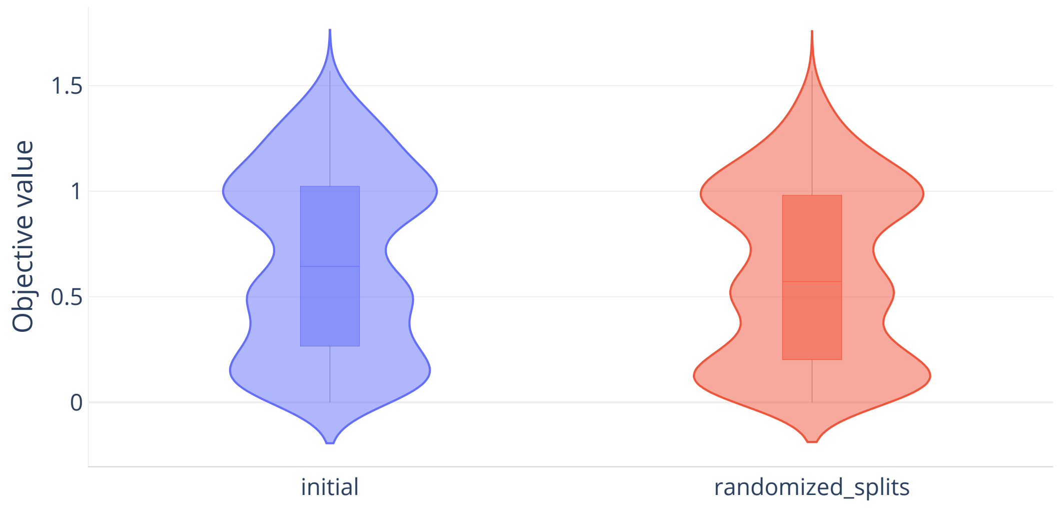


Figure 4.6: Impact of randomizing splits on objective value.

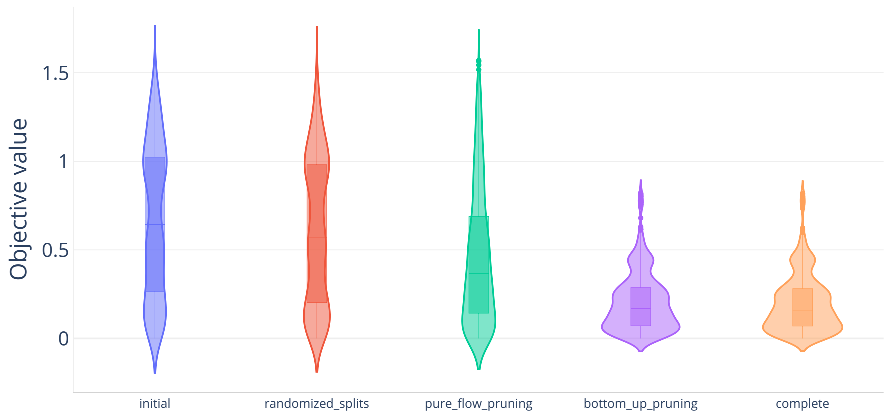
4.1.5 Overall

Lastly, we present the overall effects of each version of the constructive heuristic, for each analyzed variable. The comparison can be seen in Figure 4.7. It's clear from Figure 4.7a that bottom-up pruning accounts for most of the improvements in the objective value. For training and test accuracies, depicted in Figures 4.7b and 4.7c respectively, randomized splits appear to have the most impact, although the difference between versions is not as pronounced.

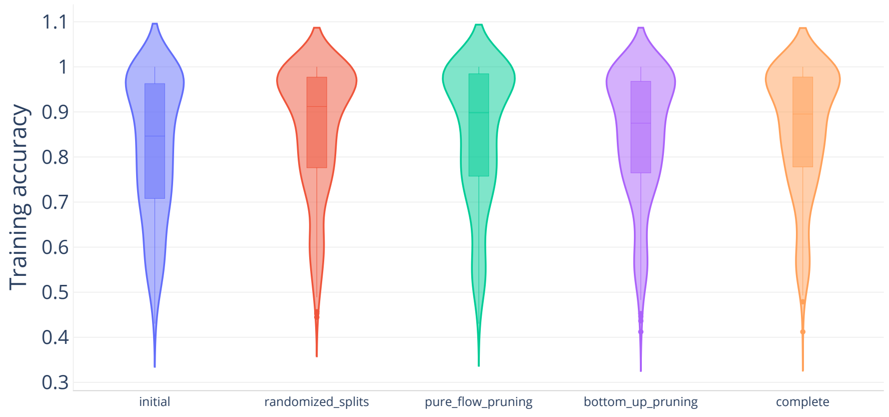
For the complete version of the heuristic, which includes all improvements, the effects on each metric follows the combined effects of all versions, as expected. Pairwise Wilcoxon rank-sum tests also indicate that these effects are statistically significant (p-values < 0.001) when compared with the initial version of the heuristic.

4.2 DNF Learning

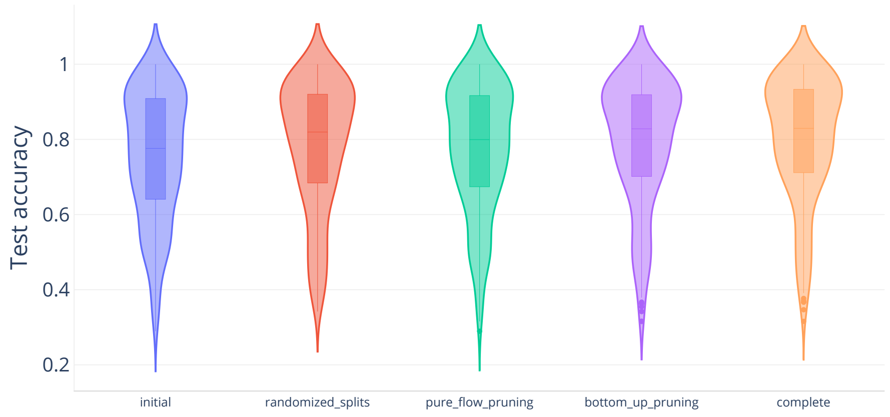
In this section, we analyze the classification performance of decision diagrams and trees for different uniformly random s-terms k-DNFs, using the constructive heuristic approach described in Chapter 3. We also present a fragmentation analysis comparing decision diagrams and trees.



(a) Objective value.



(b) Training accuracy.



(c) Test accuracy.

Figure 4.7: Impact of each heuristic version over all experiment configurations.

4.2.1 Classification Performance

For the performance experiments, we evaluated 12 s-terms k-DNFs, with fixed $k = 6$. The formulas were randomly generated with d variables, for $d = \{8, 16, 32, 64\}$, and terms count $s = \{3, 4, 5\}$. The classification task is to learn the given DNF function from a limited set of examples. For each DNF, a data set X of learning samples (x, y) was built from 1,000 balanced samples drawn from the uniform distribution, so that the feature vectors $x = (x_1, \dots, x_d)$ represent an assignment of binary variables and the target variable y is the value of the respective DNF function evaluated on this assignment. The data set X is split into training and test sets with respective proportions of 90% and 10%. The learned decision trees used the skeleton 1-2-4-8-16-32-64-128, while the diagrams used the skeleton 1-2-4-...-4 with 63 layers of at most four nodes, starting from the third layer. Both models have thus a maximum of 255 internal nodes (including the root), so as to guarantee that the models fit the random DNFs.

The constructive heuristic used pure flow pre-pruning and randomized splits, but no bottom-up pruning. This choice was made to allow for the models to better fit the DNFs without having to optimize for simplicity. We used a set of 10 seeds $\{0, 1, \dots, 9\}$ for the random number generator, affecting the data set train/test split and the randomized feature split decisions. The experiment was run five times for each configuration. The measured outcomes were the training error and test error.

Table 4.2 shows the experiment results. For small uniformly random k-DNFs, decision diagrams appear to yield better classification performance on average when compared to trees, when learning from uniformly distributed random samples.

Figure 4.8 plots the distribution of training and test error (note that, since there is no bottom-up pruning, training error, and objective value have the exact same distribution). Training accuracy is significantly improved when using the decision diagram model, but generalization is not improved as much.

4.2.2 Fragmentation

For this fragmentation analysis, we used the same setup for the classification performance experiment. To quantify overall fragmentation in a model, we compute the *support* of each node, defined as the percentage of samples that flow through that node. We filter out the root, which always has 100% support, and leaf nodes, whose (true) support is dependent on the class distri-

Table 4.2: Classification performance for learning k-DNFs

			Training error (avg.)		Test error (avg.)	
6-DNF	Size	s	diagram	tree	diagram	tree
dnf01	8	3	2.3	3.6	0.8	0.9
dnf02	8	4	6.8	8.3	0.9	1.1
dnf03	8	5	4.7	11.9	0.7	1.3
dnf04	16	3	0.1	13.1	0.6	2.1
dnf05	16	4	0.2	15.8	3.1	6.0
dnf06	16	5	0.2	27.8	5.5	7.9
dnf07	32	3	0.0	5.4	2.7	2.6
dnf08	32	4	0.0	15.0	7.2	8.9
dnf09	32	5	21.1	22.3	13.0	13.7
dnf10	64	3	0.0	9.9	3.7	6.7
dnf11	64	4	0.0	13.1	8.4	12.4
dnf12	64	5	11.4	7.0	14.0	16.8

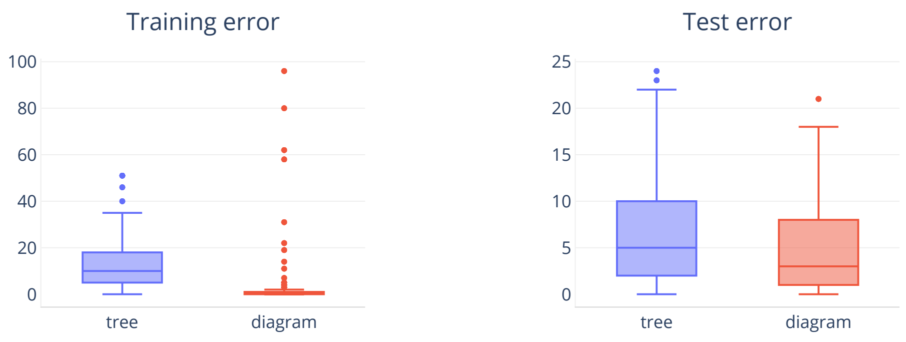
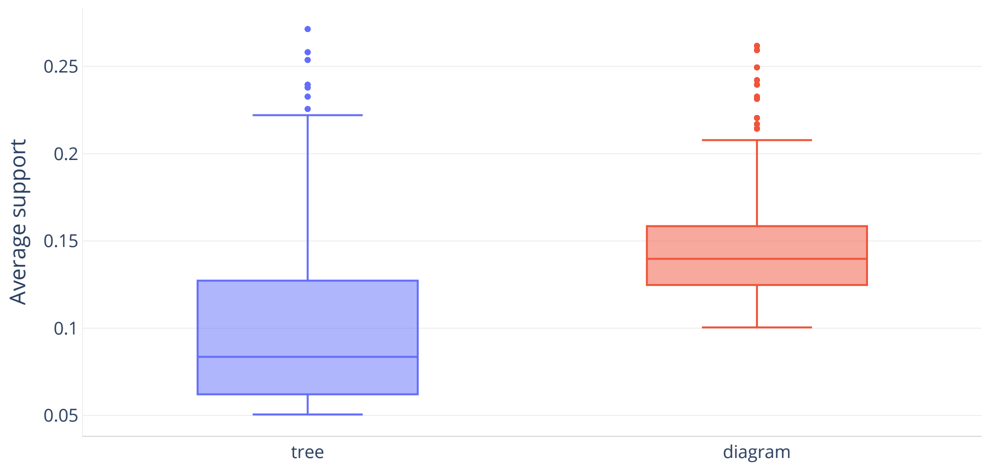


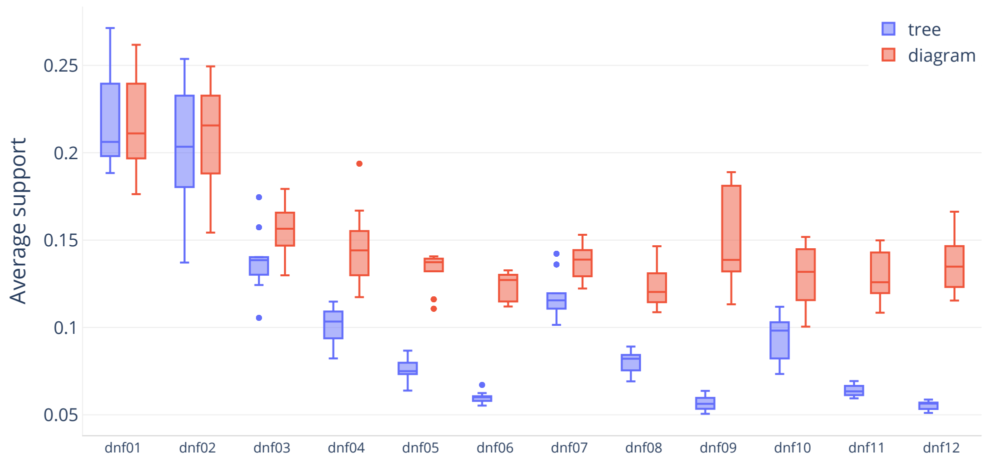
Figure 4.8: Distribution of training and test error for learning k-DNFs.

bution for the problem at hand. The quantity we measure is then the average internal node support for each learned model.

Figure 4.9 presents the results. In Figure 4.9a, the overall effect of using a diagram on the fragmentation level is already visible. The average support per DNF, as depicted in Figure 4.9b, shows the effect more clearly. For several DNFs, the support is significantly lower – and thus the fragmentation higher – in the decision tree model.



(a) Average support for all configurations.



(b) Average node support per DNF.

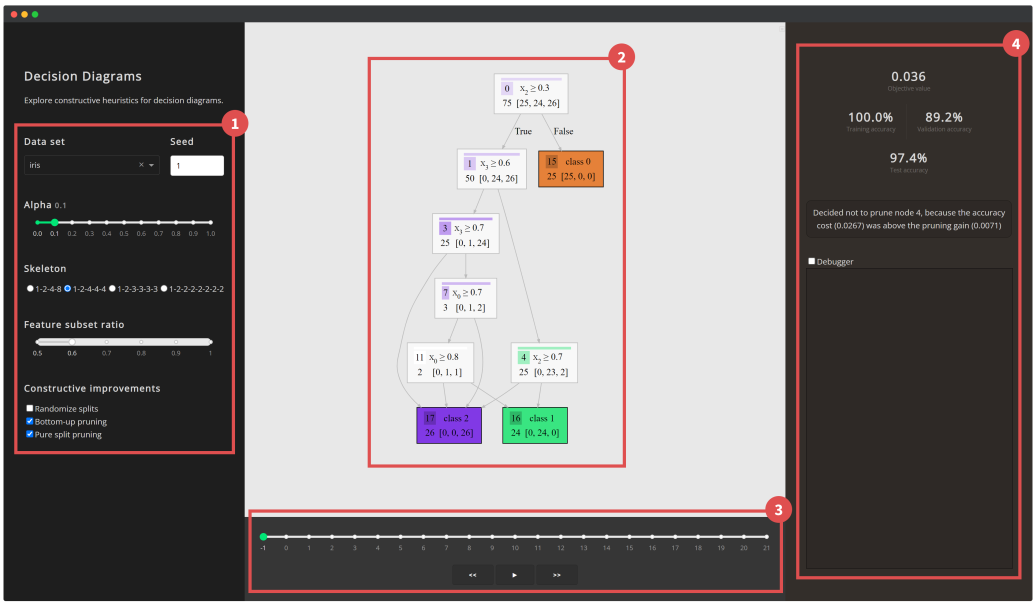
Figure 4.9: Distribution of average support per internal node, excluding root and leaves. The support of a node is defined as the percentage of samples that flow through that node. The average of node support for a model is a measure of its fragmentation (low support means high fragmentation).

5
Explorer Tool

In this chapter, we present a tool developed to debug the decision diagram constructive algorithm and analyze the proposed improvements[1]. Besides visualizing the learned diagram with split and merge decisions for a particular data set, skeleton, and α hyperparameter, the tool allows constructive improvements to be individually activated or deactivated, log messages to be shown in a debug panel, and all constructive steps to be replayed.

5.1
Overview

The explorer tool is a web application developed using the Dash framework[2] in Python. Its main objective is to provide a visual debugging experience for the decision diagram constructive approaches explored in this study. The interface is displayed in Figure 5.1.


Decision Diagrams

Explore constructive heuristics for decision diagrams.

Data set: iris

Seed: 1

Alpha 0.1

Skeleton: 1-2-4-8 [✓ 1-2-4-4-4] 1-2-3-3-3-3 1-2-2-2-2-2-2-2

Feature subset ratio: 0.6

Constructive improvements

- [ ] Randomize splits
- [x] Bottom-up pruning
- [x] Pure split pruning

0.036 Objective value

100.0% Training accuracy

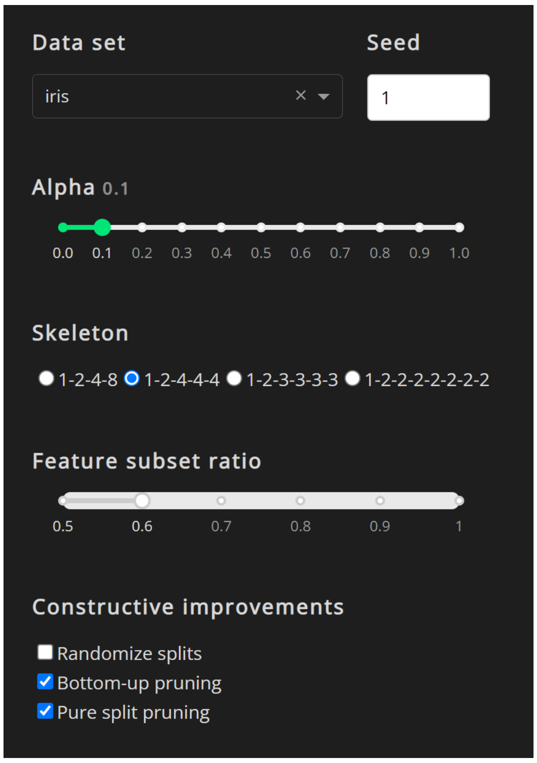
89.2% Validation accuracy

97.4% Test accuracy

Decided not to prune node 4, because the accuracy cost (0.0267) was above the pruning gain (0.0071)

- [ ] Debugger

Figure 5.1: Overview of the explorer tool GUI showing (1) the setup panel, (2) the diagram visualization, (3) the timeline, and (4) the information panel.



Data set: iris

Seed: 1

Alpha 0.1

Skeleton

1-2-4-8 [✓ 1-2-4-4-4] 1-2-3-3-3-3 1-2-2-2-2-2-2-2

Feature subset ratio: 0.5 – 0.6

Constructive improvements

- ☐ Randomize splits
- ☑ Bottom-up pruning
- ☑ Pure split pruning

Figure 5.2: The setup panel.

5.2 Setup Panel

The setup panel is displayed in more detail in Figure 5.2. It provides components for the selection of all required parameters for the construction of a decision diagram. Every time a configuration is modified, the constructive heuristic runs again and the diagram is updated.

The data sets are taken from a pre-configured directory, where they must be stored as CSV files in the usual manner (rows represent samples, with a column for each feature and a column for the target class). It is not possible to define through the interface a custom split of training, validation, and test sets, which are fixed in proportions of 50%, 25%, and 25%, respectively. The seed value affects this data split, as well as the randomization of feature splits if this improvement is activated.

5.3 Diagram Visualization

Figure 5.3 details the diagram visualization. Each internal node displays its respective index and the feature split in the form $x_i \geq a$, where x_i is

[1]The explorer is available in the public repository for the Optimal Decision Diagram (ODD) algorithm in <https://github.com/vidalt/Decision-Diagrams/tree/explorer-tool>.

[2]<https://dash.plotly.com>

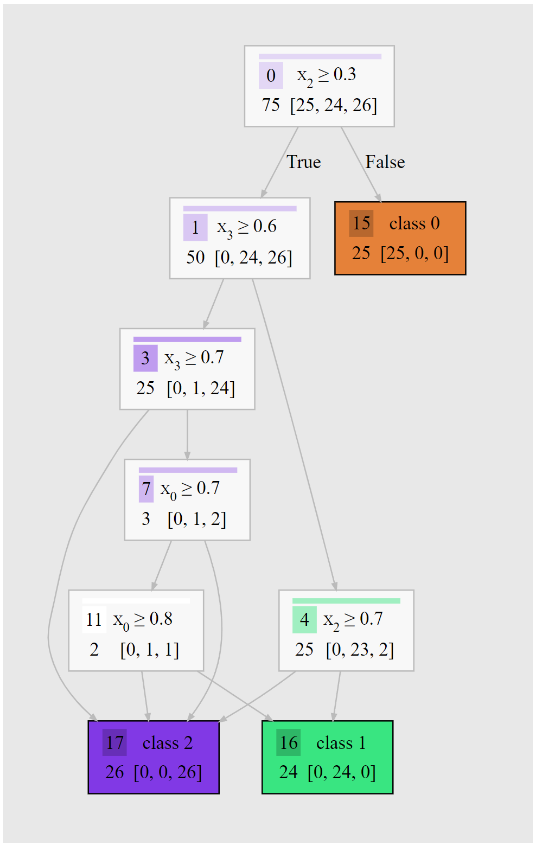


Figure 5.3: The diagram visualization.

the feature being split and a is the split value. Terminal nodes display their corresponding class instead of a split. Both types of node display a row in the form S_t $[S_0, \ldots, S_{c-1}]$, where S_t is the number of samples arriving at the node, c is the number of classes in the data set, and each S_i represents the number of samples of class $i \in [0, \ldots, c-1]$ that arrive at the node.

The color of the leaf nodes represents their respective class. In the internal nodes, the highlight color has the hue of its majority class (or white, if the samples are equally distributed among the classes), while its saturation is proportional to the percentage of samples from the majority class. This setup helps visualize how the class distribution in the internal nodes changes after each split and merge.

5.4 Timeline

The timeline component allows each constructive step to be replayed and analyzed. The step description is provided in the information panel (see Section 5.5). To control the timeline, it is possible to directly select the desired step by clicking on its number, use the previous or next step buttons to go backward or forwards by a single step, or use the play button to reproduce the

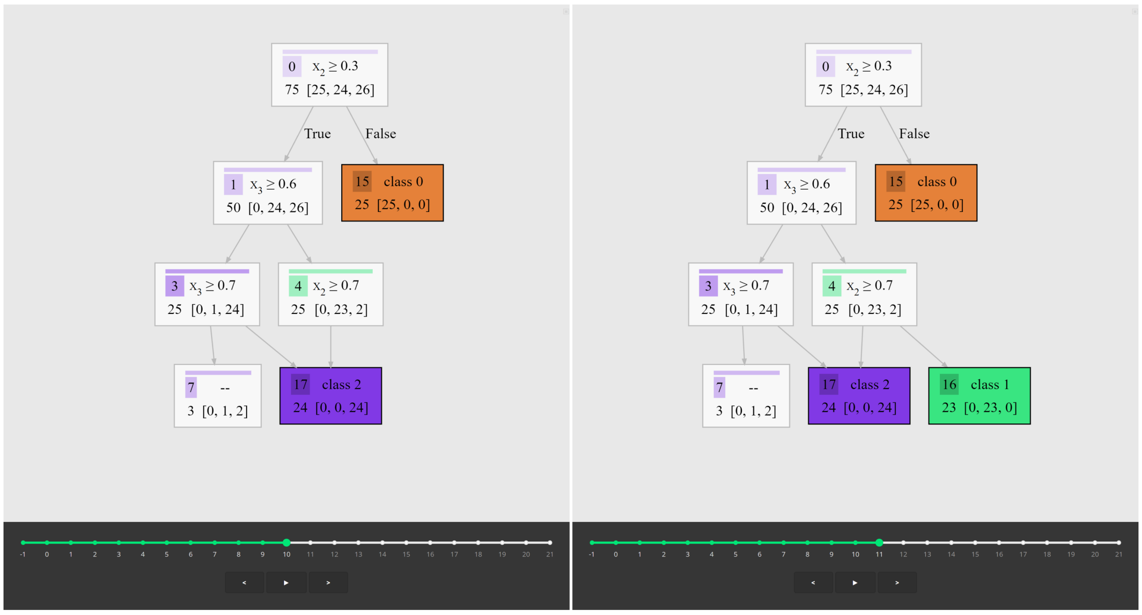
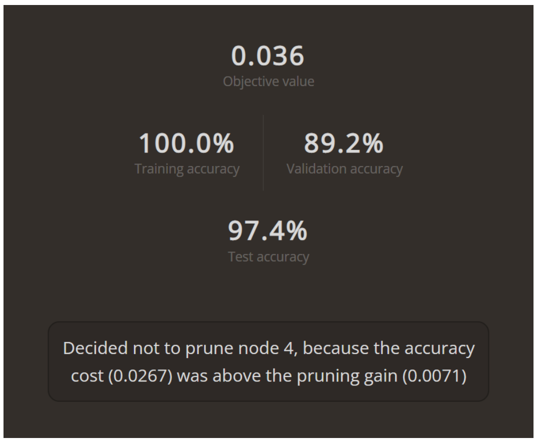


Figure 5.4: Example of the timeline component in the context of a pre-pruning constructive step.


0.036
Objective value

100.0%
Training accuracy

89.2%
Validation accuracy

97.4%
Test accuracy

Decided not to prune node 4, because the accuracy cost (0.0267) was above the pruning gain (0.0071)

Figure 5.5: The information panel.

constructive algorithm steps in sequence automatically.

A static example of the timeline in action is shown in Figure 5.4. Between steps 10 and 11 of this example, the right edge departing node 4 was connected directly to the newly created leaf representing class 1, because of the pure flow pre-pruning constructive improvement described in Section 3.3.

5.5 Information Panel

Finally, the rightmost panel displays information about the constructed diagram and the algorithm steps. As shown in Figure 5.5, the panel provides basic classification metrics for the diagram – the objective value and the training, validation, and test accuracies. It also displays the description for the last selected step, including details about the decisions. Not shown in the figure is the debugger panel, which displays messages from a debug logger.

6
Conclusion

We conclude the study by discussing its contributions and limitations. We also point to future research paths.

6.1
Our Contributions

We implemented and analyzed improvements for the heuristic construction of decision diagrams, with particular attention to the context of the Optimal Decision Diagram (ODD) algorithm (FLORIO et al., 2022). In particular, we proposed randomizing splits, pruning sample flows consisting of a single class and pruning nodes that do not contribute to the ODD objective value. The most prominent impact originated in the pruning approaches, greatly reducing the objective value of the resulting models. In the context of the ODD algorithm, this has the potential of translating into better warm starts for the *mixed-integer programming* (MILP) solver.

We also studied the expressiveness of decision diagrams when compared to decision trees, by fitting boolean functions in *disjunctive normal form* (DNFs). For small s-term k-DNF functions, decision diagrams yield better classification performance. Additionally, diagrams have less overall fragmentation of samples in their internal nodes.

Finally, we presented a web application for visually exploring the constructive algorithm analyzed in this study. We expect that further analyses and improvements can be made with the help of this tool.

6.2
Research Limitations

Decision diagrams, like decision trees, can be applied to both supervised classification and regression tasks. Our study focuses solely on the classification task and is thus an incomplete view of decision diagrams as machine learning models.

The experimental configurations have limited scope, owing to constraints in time and resources. For both the improvements and DNF experiments, the breadth of the setup could be improved in terms of different topologies (i.e., studying the improvements in deep diagram skeletons with small maximum width), as well as a higher number of random seeds and α hyperparameters for more complete coverage.

6.3 Future Research

For future work, a possible line of research is to further evaluate constructive heuristic improvements. For instance, the pure flow pruning method could be softened to allow some impurity, different randomization strategies could be considered such as randomizing merges, and metaheuristic algorithms could be applied to the problem of building a diagram. In this context, the improvement impacts could also be directly evaluated by solving the MILP formulation of the ODD algorithm and analyzing the effects on the number of optimal solutions found and in other metrics of interest, such as the resulting fragmentation level of the diagram.

Another interesting line of research would be studying decision diagrams in the context of regression tasks. To the best of our knowledge, there are currently no studies for decision diagrams that follow this path. Finally, a proper study on the interpretability of decision diagrams would be beneficial for the use of this kind of model in high-stakes decision-making.

7 Bibliography

AGLIN, G.; NIJSSEN, S.; SCHAUS, P. Learning Optimal Decision Trees Using Caching Branch-and-Bound Search. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 34, n. 04, p. 3146–3153, 2020.

ALI, K. M.; PAZZANI, M. J. Error Reduction Through Learning Multiple Descriptions. **Machine Learning**, v. 24, n. 3, p. 173–202, 1996.

ANDERSEN, H. R.; HADZIC, T.; HOOKER, J. N.; TIEDEMANN, P. A Constraint Store Based On Multivalued Decision Diagrams. In: **International Conference on Principles and Practice of Constraint Programming**. Berlin, Heidelberg: Springer-Verlag, 2007. p. 118–132.

BAHL, L.; BROWN, P.; SOUZA, P. de; MERCER, R. A Tree-Based Statistical Language Model For Natural Language Speech Recognition. **IEEE Transactions on Acoustics, Speech, and Signal Processing**, v. 37, n. 7, p. 1001–1008, 1989.

BEHLE, M. **Binary Decision Diagrams and Integer Programming**. Thesis (Doctor in Philosophy) — Saarland University, 2007.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. V.; HOOKER, J. **Decision Diagrams For Optimization**. Cham: Springer, 2016.

BERTSIMAS, D.; DUNN, J. Optimal Classification Trees. **Machine Learning**, Springer US, v. 106, n. 7, p. 1039–1082, 2017.

BIXBY, R. E. A Brief History of Linear and Mixed-Integer Programming Computation. **Documenta Mathematica**, p. 107–121, 2012.

BLUM, A.; FURST, M.; JACKSON, J.; KEARNS, M.; MANSOUR, Y.; RUDICH, S. Weakly Learning DNF And Characterizing Statistical Query Learning Using Fourier Analysis. In: **Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing**. New York: ACM Press, 1994. p. 253–262.

BREIMAN, L. Bagging Predictors. **Machine Learning**, v. 24, n. 2, p. 123–140, 1996.

BREIMAN, L. Random Forests. **Machine Learning**, v. 45, n. 1, p. 5–32, 2001.

BREIMAN, L.; FRIEDMAN, J. H.; OLSHEN, R. A.; STONE, C. J. **Classification and Regression Trees**. Boca Raton: Chapman & Hall, 1984.

BRESLOW, L. A.; AHA, D. W. Simplifying decision trees: A survey. **The Knowledge Engineering Review**, v. 12, n. 01, p. 1–40, 1997.

Bryant. Graph-Based Algorithms for Boolean Function Manipulation. **IEEE Transactions on Computers**, C-35, n. 8, p. 677–691, 1986.

BRYANT, R. E. Symbolic Boolean Manipulation With Ordered Binary-Decision Diagrams. **ACM Computing Surveys**, v. 24, n. 3, p. 293–318, 1992.

CABODI, G.; CAMURATI, P. E.; IGNATIEV, A.; MARQUES-SILVA, J.; PALENA, M.; PASINI, P. Optimizing Binary Decision Diagrams for Interpretable Machine Learning Classification. In: **2021 Design, Automation & Test in Europe Conference & Exhibition**. Grenoble: IEEE, 2021. p. 1122–1125.

CARRIZOSA, E.; MOLERO-RÍO, C.; MORALES, D. R. Mathematical Optimization In Classification And Regression Trees. **Top**, Springer Berlin Heidelberg, v. 29, n. 1, p. 5–33, 2021.

CARVALHO, D. V.; PEREIRA, E. M.; CARDOSO, J. S. **Machine Learning Interpretability: A Survey On Methods And Metrics**. [S.l.]: MDPI AG, 2019.

CASTRO, M. P.; PIACENTINI, C.; CIRE, A. A.; BECK, J. C. Relaxed BDDS: An Admissible Heuristic For Delete-Free Planning Based On A Discrete Relaxation. In: **Proceedings of the International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2019. v. 29, p. 77–85.

CHOROWSKI, J.; ZURADA, J. M. Extracting Rules From Neural Networks As Decision Diagrams. **IEEE Transactions on Neural Networks**, v. 22, n. 12 PART 2, p. 2435–2446, 2011.

COSTA, V. G.; PEDREIRA, C. E. Recent Advances In Decision Trees: An Updated Survey. **Artificial Intelligence Review**, v. 56, n. 5, p. 4765–4800, 2022.

CUTLER, A.; ZHAO, G. PERT-Perfect Random Tree Ensembles. **Computing Science and Statistics**, v. 33, 2001.

DEMIROVIĆ, E.; LUKINA, A.; HEBRARD, E.; CHAN, J.; BAILEY, J.; LECKIE, C.; RAMAMOHANARAO, K.; STUCKEY, P. J. MurTree: Optimal Classification Trees via Dynamic Programming and Search. **ArXiv**, abs/2007.12652, 2020.

DIETTERICH, T. G. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. **Machine Learning**, v. 40, n. 2, p. 139–157, 2000.

DIETTERICH, T. G. Ensemble Methods in Machine Learning. In: **Multiple Classifier Systems**. Berlin: Springer, 2000. v. 1857, p. 1–15.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. 2017. Available at: <http://archive.ics.uci.edu/ml>.

ELOMAA, T.; KAARIAINEN, M. An Analysis of Reduced Error Pruning. **Journal of Artificial Intelligence Research**, v. 15, 2011.

FLORIO, A. M.; MARTINS, P.; SCHIFFER, M.; SERRA, T.; VIDAL, T. Optimal Decision Diagrams for Classification. **Thirty-Seventh AAAI Conference on Artificial Intelligence**, 2022.

FRIEDMAN, J. H. Greedy Function Approximation: A Gradient Boosting Machine. **The Annals of Statistics**, v. 29, n. 5, 2001.

FÜRNKRANZ, J.; GAMBERGER, D.; LAVRA, N. Foundations of Rule Learning. In: **Cognitive Technologies**. [S.l.: s.n.], 2012.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely Randomized Trees. **Machine Learning**, v. 63, n. 1, p. 3–42, 2006.

GORISHNIY, Y.; RUBACHEV, I.; KHRULKOV, V.; BABENKO, A. Revisiting Deep Learning Models for Tabular Data. **Advances in Neural Information Processing Systems**, v. 34, 2021.

GRINSZTAJN, L.; OYALLON, E.; VAROQUAUX, G. Why Do Tree-Based Models Still Outperform Deep Learning On Tabular Data? **NeurIPS 2022 Datasets and Benchmarks Track**, 2022.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. New York: Springer, 2001. (Springer Series in Statistics).

HU, H.; HUGUET, M.-J.; SIALA, M. Optimizing Binary Decision Diagrams with MaxSAT for classification. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 36, 2022.

HYAFIL, L.; RIVEST, R. L. Constructing Optimal Binary Decision Trees Is NP-Complete. **Information Processing Letters**, Elsevier, v. 5, n. 1, p. 15–17, 1976.

IGNATOV, D.; IGNATOV, A. Decision Stream: Cultivating Deep Decision Trees. **Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI**, v. 29, p. 905–912, 2018.

JACKSON, J.; LEE, H.; SERVEDIO, R.; WAN, A. Learning Random Monotone DNF. In: **Proceedings of the 11th International Workshop, APPROX 2008, and 12th International Workshop, RANDOM 2008 on Approximation, Randomization and Combinatorial Optimization: Algorithms and Techniques**. Berlin: Springer-Verlag, 2008. v. 14, p. 483–497.

JACKSON, J. C. An Efficient Membership-Query Algorithm for Learning DNF with Respect to the Uniform Distribution. **Journal of Computer and System Sciences**, v. 55, n. 3, p. 414–440, 1997.

KOHAVI, R. Bottom-Up Induction Of Oblivious Read-Once Decision Graphs. **Lecture Notes in Computer Science**, v. 784, p. 154–169, 1994.

KOHAVI, R.; LI, C.-H. Oblivious Decision Trees Graphs and Top-Down Pruning. In: **Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. (IJCAI'95), p. 1071–1077.

LAI, Y.; LIU, D.; WANG, S. Reduced Ordered Binary Decision Diagram With Implied Literals: A New Knowledge Compilation Approach. **Knowledge and Information Systems**, v. 35, p. 665–712, 2013.

LAKKARAJU, H.; BACH, S. H.; LESKOVEC, J. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York: ACM, 2016. p. 1675–1684.

LANGE, J.-H.; SWOBODA, P. Efficient Message Passing for 0-1 ILPs with Binary Decision Diagrams. **Proceedings of the 38th International Conference on Machine Learning**, v. 139, p. 6000–6010, 2021.

LEE, C. Y. Representation of Switching Circuits by Binary-Decision Programs. **Bell System Technical Journal**, v. 38, n. 4, p. 985–999, 1959.

LOH, W.-Y. Fifty Years of Classification and Regression Trees. **International Statistical Review**, v. 82, n. 3, p. 329–348, 2014.

MAHONEY, J. J.; MOONEY, R. J. **Initializing ID5R with a Domain Theory: Some Negative Results**. Austin, 1991.

MESSENGER, R.; MANDELL, L. A Modal Search Technique for Predictibe Nominal Scale Multivariate Analys. **Journal of the American Statistical Association**, v. 67, n. 340, p. 768, 1972.

MINGERS, J. An Empirical Comparison of Pruning Methods for Decision Tree Induction. **Machine Learning**, v. 4, n. 2, p. 227–243, 1989.

MOONEY, P. **2022 Kaggle Machine Learning & Data Science Survey**. Kaggle, 2022. Available at: <https://kaggle.com/competitions/kaggle-survey-2022>.

MORGAN, J. N.; SONQUIST, J. A. Problems in the Analysis of Survey Data, and a Proposal. **Journal of the American Statistical Association**, v. 58, n. 302, p. 415–434, 1963.

NARODYTSKA, N.; IGNATIEV, A.; PEREIRA, F.; MARQUES-SILVA, J. Learning Optimal Decision Trees with SAT. In: **Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence**. Richland: International Joint Conferences on Artificial Intelligence Organization, 2018. p. 1362–1368.

NIEUWENHUIS, R. A. I.; A, O.; E., R.-C.; V, M.-E. A New Look at BDDs for Pseudo-Boolean Constraints. **Journal of Artificial Intelligence Research**, v. 45, p. 443–480, 2012.

OLIVEIRA, A. L.; SANGIOVANNI-VINCENTELLI, A. Using The Minimum Description Length Principle To Infer Reduced Ordered Decision Graphs. **Machine Learning**, v. 25, n. 1, p. 23–50, 1996.

OLIVER, J. J. Decision Graphs - An Extension of Decision Trees. **Fourth International Workshop on Artificial Intelligence and Statistics**, v. 173, p. 343–350, 1993.

PAGALLO, G. Learning DNF by Decision Trees. In: **Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989. (IJCAI'89), p. 639–644.

PEREZ, G.; RÉGIN, J.-C. Efficient Operations On MDDs For Building Constraint Programming Models. In: **Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence**. RIchland: International Joint Conferences on Artificial Intelligence Organization, 2015. p. 173–189.

PITT, L.; VALIANT, L. G. Computational Limitations On Learning From Examples. **Journal of the ACM**, v. 35, n. 4, p. 965–984, 1988.

PLATT, J. C.; CRISTIANINI, N.; SHAWE-TAYLOR, J. Large Margin DAGs for Multiclass Classification. **Advances in Neural Information Processing Systems**, p. 547–553, 2000.

QUINLAN, J. Simplifying Decision Trees. **International Journal of Man-Machine Studies**, v. 27, n. 3, p. 221–234, 1987.

QUINLAN, J. R. Induction of Decision Trees. **Machine Learning**, v. 1, n. 1, p. 81–106, 1986.

QUINLAN, J. R. **C4.5: Programs for Machine Learning**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

RUDIN, C. Stop Explaining Black Box Machine Learning Models For High Stakes Decisions And Use Interpretable Models Instead. **Nature Machine Intelligence**, v. 1, n. 5, p. 206–215, 2019.

RUDIN, C.; CHEN, C.; CHEN, Z.; HUANG, H.; SEMENOVA, L.; ZHONG, C. Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges. **Statistics Surveys**, v. 16, 2022.

SANNER, S.; UTHER, W. T. B.; DELGADO, K. V. Approximate Dynamic Programming With Affine ADDs. In: **AAMAS**. [S.l.: s.n.], 2010. p. 1349–1356.

SELLIE, L. Learning Random Monotone DNF Under the Uniform Distribution. In: **Annual Conference Computational Learning Theory**. [S.l.: s.n.], 2008.

SELLIE, L. Exact Learning of Random DNF Over the Uniform Distribution. In: **Proceedings of the forty-first annual ACM symposium on Theory of computing**. New York, NY, USA: ACM, 2009. p. 45–54.

SERRA, T. Enumerative Branching With Less Repetition. In: **International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research**. Cham: Springer, 2020. p. 399–416.

SHOTTON, J.; NOWOZIN, S.; SHARP, T.; WINN, J.; KOHLI, P.; CRIMINISI, A. Decision Jungles: Compact and Rich Models for Classification. **Advances in Neural Information Processing Systems**, p. 1–9, 2013.

SHWARTZ-ZIV, R.; ARMON, A. Tabular Data: Deep Learning is Not All You Need. **Information Fusion**, v. 81, p. 84–90, 2022.

Tin Kam Ho. The Random Subspace Method For Constructing Decision Forests. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 20, n. 8, p. 832–844, 1998.

TOMSETT, R.; BRAINES, D.; HARBORNE, D.; PREECE, A.; CHAKRABORTY, S. Interpretable to Whom? A Role-based Model for Analyzing Interpretable Machine Learning Systems. **ArXiv**, abs/1806.07552, 2018.

VALIANT, L. G. A Theory of the Learnable. **Communications of the ACM**, v. 27, n. 11, p. 1134–1142, 1984.

VERBEURGT, K. Learning DNF Under the Uniform Distribution in Quasi-Polynomial Time. In: **Proceedings of the Third Annual Workshop on Computational Learning Theory**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990. (COLT '90), p. 314–326.

VERHAEGHE, H.; LECOUTRE, C.; SCHAUS, P. Compact-MDD: Efficiently Filtering (s)MDD Constraints With Reversible Sparse Bit-Sets. In: **Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence**. Richland: Interna- tional Joint Conferences on Artificial Intelligence Organization, 2018. p. 1383–1389.

VIKTOR, H. L.; CLOETE, I. Extracting DNF Rules From Artificial Neural Networks. In: **From Natural to Artificial Neural Computation**. Berlin: Springer, 1995. v. 930, p. 611–618.

VILALTA, R.; BLIX, G.; RENDELL, L. Global Data Analysis and the Fragmentation Problem in Decision Tree Induction. In: **Machine Learning: ECML-97**. Berlin: Springer, 1997. v. 1224, p. 312–326.