Optimizing Binary Decision Diagrams for Interpretable Machine Learning Classification

Gianpiero Cabodi[*], Paolo E. Camurati[*], Alexey Ignatiev[†], Joao Marques-Silva[‡], Marco Palena[*] and Paolo Pasini[*]
[*]*DAUIN*, *Politecnico di Torino*, Turin, IT
{gianpiero.cabodi,paolo.camurati,marco.palena,paolo.pasini}@polito.it
[†]*Monash University*, Melbourne, AU
alexey.ignatiev@monash.edu
[‡]*ANITI*, *University of Toulouse*, Toulouse, FR
joao.marques-silva@univ-toulouse.fr

***Abstract*—Motivated by the need to understand the behaviour of complex machine learning (ML) models, there has been recent interest in learning optimal (or sub-optimal) decision trees (DTs). This interest is explained by the fact that DTs are widely regarded as interpretable by human decision makers. An alternative to DTs are Binary Decision Diagrams (BDDs), which can be deemed interpretable. Compared to DTs, and despite a fixed variable order, BDDs offer the advantage of more compact representations in practice, due to node sharing. Moreover, there is also extensive experience in the efficient manipulation of BDDs. Our work proposes preliminary inroads in two main directions: (a) proposing a SAT-based model for computing a decision tree as the smallest Reduced Ordered Binary Decision Diagram, consistent with given training data; and (b) exploring heuristic approaches for deriving sub-optimal (i.e., not minimal) ROBDDs, in order to improve the scalability of the proposed technique. The heuristic approach is related to recent work on using BDDs for classification. Whereas previous works addressed size reduction by general logic synthesis techniques, our work adds the contribution of generalized cofactors, that are a well-known compaction technique specific to BDDs, once a care (or equivalently a don't care) set is given. Preliminary experimental results are also provided, proposing a direct comparison between optimal and sub-optimal solutions, as well as an evaluation of the impact of the proposed size reduction steps.**

***Index Terms*—Binary Decision Diagrams, SAT, Classification**

I. Introduction

Social acceptance of Machine Learning (ML) entails going beyond the classical black-box model, where even the designer cannot explain the conclusions that have been reached. The area of *eXplainable Artificial Intelligence* (XAI) responds to the social right to explanation. In the context of binary classification, ML models can be seen as representations of Boolean functions, mapping a set of binary features to one of two classes. In such a context, (binary) decision trees represent a natural choice to support explainability, provided their size does not hamper their understandability by a human. Binary Decision Diagrams (BDDs) are a well-established data structure known to provide compact representations of Boolean functions. Given their succinctness, BDDs make for very explainable models for binary classification problems. This paper addresses the following problems, given a consistent dataset for binary classification:

- learn the optimal BDD model, i.e., the smallest BDD that exhibits 100% accuracy on the training data. This paper proposes a novel SAT model for deciding the existence of structurally sound BDDs, consistent with training data. An iterative SAT-based procedure is used to learn the optimal BDD. This technique works well with small problems, but doesn't scale to bigger ones;
- develop heuristic techniques based on operators typical of BDDs like the cofactor to improve scalability, though the resulting BDD is not necessarily minimum-size;
- compare suboptimal and optimal results, when available, in terms of BDD size and CPU time to prove the feasibility and the usefulness of the heuristic approach.

A. Related Works

The complexity of learning optimal decision trees is properly established in the literature [1], [2]. The problem is categorized as NP-hard, thus infeasible in practice [3], for many applications. Regardless of complexity, earlier attempts at learning optimal decisions exist, both using constraint programming [4] and SAT-based approaches [5], [6]. A heuristic approach towards BDD-based classifiers is presented in [7].

II. Preliminaries

A. Classification Problems

We follow the standard notation used in classification problems. A set of binary features $\mathcal{F} = \{x_1, \ldots, x_K\}$, $|\mathcal{F}| = K$, takes a value in $\mathbb{B} = \{0, 1\}$. We limit ourselves to classification problems with two classes $\mathcal{K} = \{\oplus, \ominus\}$, where we associate $\oplus$ with 1 and $\ominus$ with 0. If necessary, one-hot-encoding is assumed for non-binary categorical features. Non-binary features are beyond the scope of this paper. We start from a given training data (or examples) $\mathcal{E} = \{\epsilon_1, \ldots, \epsilon_M\}$. $\mathcal{E}$ is partitioned into $\mathcal{E}^+$ and $\mathcal{E}^-$, as we consider binary classification, denoting examples classified as positive and as negative, respectively. As all features are binary, a literal on feature x_r, represented as x_r ($\neg x_r$), denotes that the feature takes value 1 (0). An example $\epsilon_q \in \mathcal{E}$ is represented as a $2-$tuple $(\mathcal{L}_q, c_q)$, where $\mathcal{L}_q$ denotes the set of literals associated with the value of features in ϵ_q and $c_q \in \mathbb{B}$ is the class (or *prediction*) to which the example belongs: $c_q = 1$ (0) if $\epsilon_q \in \mathcal{E}^+$ $(\mathcal{E}^-)$.

B. Boolean formulae, functions and BDDs

We use standard notation and definitions for Boolean formulae and functions. A *cube* (a *clause*) is a conjunction (a disjunction) of literals. A Boolean formula is in *Conjunctive Normal Form* (CNF) iff it is a conjunction of clauses. A truth assignment for a Boolean formula F is a function $\tau : \mathbb{B}^n \to \mathbb{B}$ that maps variables in F to truth values.

Let $F(x_1, \ldots, x_n) : \mathbb{B}^n \to \mathbb{B}$ be a Boolean function with n variables. Its *onset* (*offset*) is the set of truth assignments such that $F(x_1, \ldots, x_n) = 1$ (0). Their union is the *careset* of F. The don't care set *dcset* is the set of truth assignments such that $F(x_1, \ldots, x_n) = -$, where $-$ stands for don't care. A function F is *completely specified* iff its *dcset* is the empty set, otherwise it is *incompletely specified*. A *completely specified* function is defined by either its *onset* or its *offset*. An *incompletely specified* function is defined by two out of its *onset*, *offset* or *dcset*. A *cover* C of a completely specified function F coincides with its *onset*. If the function is incompletely specified, the cover includes the *onset* and is included in the union of *onset* and *dcset*: $onset(f) \subseteq C \subseteq onset(F) \bigcup dcset(F)$.

The *positive* and *negative cofactors* of function F with respect to variable x_i are $F_{x_i} = F(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$ and $F_{\neg x_i} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$. The *Boole expansion theorem* expresses F in terms of its positive and negative cofactors for a variable x_i: $f(x_1, \ldots, x_n) = x_i \wedge f_{x_i} \vee \neg x_i \wedge f_{\neg x_i}$. Cofactors can be *generalized* with respect to a function g: $f = g \wedge f_g \vee \neg g \wedge f_{\neg g}$. Generalized cofactors are incompletely specified by definition.

Boolean functions find non trivial use in the representation of sets. Let A be an arbitrary finite set with cardinality $|A|$. A binary encoding enc of A is an injective mapping $enc : A \to \mathbb{B}^n$. The value of n is lower bounded by $n \geq \lceil log_2(|A|) \rceil$. Whenever $n = \lceil log_2(|A|) \rceil$ the encoding is minimum-length. A popular alternative is one-hot encoding where $n = |A|$. Given a binary encoding enc, any $S \subset A$ may be represented by its characteristic function $\chi_S(x_1, \ldots, x_n)$ where $\forall (x_1, \ldots, x_n) \chi_S(x_1, \ldots, x_n) = 1$ iff $enc(s) = (x_1, \ldots, x_n) \wedge s \in S$.

Boolean functions are key in many domains: in digital design, they are used for logic synthesis, formal verification and testing. Recent extensions concern explainability, where being able to represent and manipulate Boolean functions is fundamental. Among the most popular representations, a widely-known one are *reduced-ordered Binary Decision Diagrams* (ROBDDs or simply BDDs). Given a fixed ordering for its variables, a Boolean function can be represented as a rooted, directed, acyclic graph with two sink (terminal) nodes, labelled with truth values 0 and 1. Each non-sink node is a decision node labelled with a variable and has two children nodes. The edge from a parent node to a child represents an assignment of a variable to either truth value. Variables appear in the same order on all paths from the root to the sink nodes. Isomorphic subgraphs are merged and nodes with isomorphic subgraphs as children are removed.

BDDs are canonical representations, i.e., the representation of a function is unique given an ordering of its variables. Many techniques have been developed to find BDD variable orderings that lead to compact BDDs [8], [9], but finding the best variable ordering is an NP-hard problem. For a given variable ordering, the BDD representation of a completely specified function is unique. For incompletely specified functions, however, many BDDs can be used to represent the function, each associated with a different assignment of don't cares to binary values. This paper assumes the variable ordering is fixed and one of its contributions is to address the problem of finding an assignment of don't cares that yields a small BDD representation.

III. Computing the Smallest BDD

This section develops a SAT model for computing the smallest BDD consistent with a given dataset. Some modelling ideas are adapted from [5], [10]. We envision an iterative approach, in which we use a SAT oracle to guess a BDD classifier of minimum size. At each step we construct a SAT problem asking if there exists a BDD of a given size N consistent with training data. The optimal BDD size is found in two phases. First we identify the range in which the optimum lies by performing an exponential search: starting with $N = 3$ (lower bound on the BDD size) we double N until a SAT response is found. Then a binary search is performed on the identified range to determine the minimum BDD size.

A. Notation

At each iteration we consider a BDD of size N. When referring to BDD nodes, the indices n, i ($1 \leq n, i \leq N$) will be used, 1 for the root node, and $N - 1$ and N for terminal nodes. Classes are denoted by $\ominus$ and $\oplus$. Indices k, j, l, with $1 \leq k, j, l \leq K$, will be used for features. Finally, the index m, with $1 \leq m \leq M$, will be used for instances.

B. Guessing a valid BDD

We use the following variables to guess a valid BDD:

- $a_{kn} = 1$ iff feature x_k is associated with BDD node n, with $1 \leq k \leq K$ and $1 \leq n < N - 1$.
- $p_{in} = 1$ iff node i is the parent of node n, with $1 \leq i < n \leq N \wedge i < N - 1$. There is no parent node for $n = 1$.
- $lc_{ni} = 1$ iff node n is the left child of i, with $1 \leq i < n \leq N \wedge i < N - 1$. The left node is labelled with 0.
- $rc_{ni} = 1$ iff node n is the right child of i, with $1 \leq i < n \leq N \wedge i < N - 1$. The right node is labelled with 1.
- pr_{kl}: feature x_k precedes feature x_l in ranking.

We then define the following constraints representing a valid BDD, with N nodes, and K features.

$$\sum_{1 \leq k \leq K} a_{kn} = 1 \qquad 1 \leq n < N - 1 \tag{1}$$

$$\sum_{1 \leq i < n} p_{in} \geq 1 \qquad 1 < n \leq N \tag{2}$$

$$\begin{array}{ll} p_{in} \leftrightarrow lc_{ni} \vee rc_{ni} & 1 \leq i < n, 1 < n \leq N \\ \neg lc_{ni} \vee \neg rc_{ni} & \text{as above} \end{array} \tag{3}$$

$$\begin{array}{ll} \sum_{i < n \leq N} lc_{ni} = 1 & 1 \leq i < N - 1 \\ \sum_{i < n \leq N} rc_{ni} = 1 & \text{as above} \end{array} \tag{4}$$

$$pr_{kl} \vee pr_{lk} \quad 1 \le k, l \le K, k \ne l \tag{5}$$

$$a_{kn} \wedge p_{in} \rightarrow (a_{li} \rightarrow pr_{lk}) \quad \begin{matrix} 1 \le k, l \le K \\ 1 \le i < n < N-1 \end{matrix} \tag{6}$$

$$pr_{kk} = 0 \qquad 1 \le k \le K \tag{7}$$

$$pr_{kl} \wedge pr_{lj} \rightarrow pr_{kj} \quad \begin{matrix} 1 \le k, l, j \le K \\ k \ne l, l \ne j, j \ne k \end{matrix} \tag{8}$$

The meaning of these constraints is: 1) Each (non-terminal) BDD node is associated with exactly one feature (or variable). 2) Each BDD node other than 1 has at least one parent. Node N must not have $N-1$ as its parent so we add the unit clause $\neg p_{N-1N}$. 3) The parent of a node must have that node as its left or the right child, and a node cannot be both left and right child of another node. 4) For any non-terminal node, there must exist exactly two other nodes, one of which is its left child and the other one its right child. 5) Relative order of features in ranking is consistent. 6) For nodes other than 1, $N-1$ and N, the feature assigned to the parent node must precede the feature assigned to the node in ranking. This constraint grows with $\mathcal{O}(K^2N^2)$ clauses of 4 literals each. 7) A feature cannot precede itself in ranking. 8) Precedence in ranking is transitive. This constraint grows with $\mathcal{O}(K^3)$.

C. BDD consistent with dataset

Starting from a consistent dataset, we will require that all instances are correctly classified[*]. This is achieved by ensuring that each instance is evaluated to the correct prediction along one path in the BDD and it is not evaluated to the incorrect prediction along any path in the BDD. To validate the instances in the dataset, let us define a new set of variables e_{mn} stating whether or not a node n *evaluates* instance m, i.e., it appears in the root-leaf path that is traversed while evaluating instance m in the BDD. We also add the following set of constraints:

$$e_{m1} = 1 \quad 1 \le m \le M \tag{9}$$

$$\begin{matrix} e_{mN-1} = 1 & 1 \le m \le M \text{ if prediction } c_m = 0 \\ e_{mN} = 1 & 1 \le m \le M \text{ if prediction } c_m = 1 \end{matrix} \tag{10}$$

$$e_{mN} \leftrightarrow \neg e_{mN-1} \quad 1 \le m \le M \tag{11}$$

$$e_{mn} \leftrightarrow \bigvee\nolimits_{\substack{1 \le i < n \\ 1 \le k \le K}} e_{mi} \wedge a_{ki} \wedge ec\,(n,i,m,k) \quad \begin{matrix} 1 < n \le N \\ 1 \le m \le M \end{matrix} \tag{12}$$

$$ec\,(n,i,m,k) = \begin{cases} lc_{ni}, & \text{if } \neg x_k \in \mathcal{L}_m \\ rc_{ni}, & \text{otherwise} \end{cases} \tag{13}$$

The meaning of these constraints is as follows: 9) The root node evaluates all instances. 10) Terminal node 0 (1) evaluates each instance with prediction value 0 (value 1). 11) An instance is either evaluated at terminal node 1 or 0, not at both. 12) A node n evaluates an instance m iff a parent i of that node evaluates the instance and n is either a left or right child of i in accordance with the value of the feature assigned to the parent in the instance 13) Function ec maps a tuple (n, i, m, k) to variable lc_{ni} (rc_{ni}) if feature k takes value 0 (value 1, respectively) in instance m.

IV. Classification by BDD-based coverage

This section describes a heuristic approach to a BDD-based classifier, which is motivated by scalability issues. Our approach, related to [7] (addressing a networking-based application), explores BDD-based minimization strategies known in logic synthesis of combinational circuits. Overall, we aim at finding a small cover, using the data (training) set as its *careset*. Then we exploit generalized cofactors as BDD simplification operators. The two main challenges are keeping BDD size as small as possible and generating a classifier with high accuracy.

Both aspects need to be evaluated experimentally. Since accuracy is highly related to the quality of the training set, we mainly focus on the first aspect.

A. The BDD-based classifier

The base procedure is split in two steps. We first compute the BDDs encoding the dataset $\mathcal{E}(x)$, that represents the care set of the classifier, as well as the BDDs representing the sets[†] of instances of individual classes: $\mathcal{E}^+(x)$ and $\mathcal{E}^-(x)$.

The classifier is obtained from the dataset by providing a completely specified extension of $\mathcal{E}^+$ (or $\mathcal{E}^-$ as in a binary case just one of them is enough). Let Φ be the target classifier function. Then $\Phi^+(x) = \mathcal{E}^+(x)|\mathcal{E}(x)$ and $\Phi^-(x) = \mathcal{E}^-(x)|\mathcal{E}(x)$, where the $|$ operator represents a generalized cofactor that could be either "constrain" or "restrict" a Boolean function.

B. The cover problem

The proposed operation is actually an instance of a cover problem where the target function (Φ^+) can be considered as the cover of the incompletely specified function $\mathcal{E}^+$.

Whereas [7] generates Φ^+ by a so called sequential approach, which iterates on cubes of $\mathcal{E}^+$, individually optimized, we compute it by generalized cofactor of (the BDD of) $\mathcal{E}^+$ w.r.t. $\mathcal{E}$. A fair comparison of the two approaches should be done both in terms of BDD size and of classification accuracy. Our experimental results generally lead to the conclusion that cofactors have some edge in terms of BDD size, whereas in terms of classification accuracy we are not able to come to a definite conclusion, as the final quality of the classifier seems more related to the overall quality of the dataset.

Based on the previously described approach, we have also implemented a variant interleaving the sequential cover and the cofactoring steps. The resulting procedure first computes the care set $\mathcal{E}(x)$, then it iteratively picks cubes in $\mathcal{E}^+$, it minimizes them in terms of prime implicants (as in the sequential cover), and finally it cofactors them with the careset $\mathcal{E}$. The target cover is initialized as $\Phi_0^+(x) = 0$, then it is iteratively refined as

$$\Phi_i^+(x) = \Phi_{i-1}^+(x) \vee p_i^+(x)|\mathcal{E}(x), \quad i > 0$$

where p_i^+ is a prime implicant generated by $\epsilon_i^+(x)$.

[*] Though possible, an accuracy less than 100% is beyond the scope of this work.

[†] With abuse of notation, in line with most past works on BDDs, we use the same notation for a set and its characteristic function.

V. Experimental Evaluation

We have implemented a prototype version of the proposed SAT-based approach using PySAT [11]. The heuristic BDD synthesis method has been implemented in PdTRAV [12], a state-of-the-art formal verification tool. We performed experiments on a set of benchmarks derived from [13]. Focusing on datasets for binary classification problems, a selection of datasets has been preprocessed (converting features to binary) and sub-sampled (to reduce problem size). Binarization is performed using the well-known one-hot-encoding technique. Sub-sampling is needed to reduce the search space to a much more manageable size. The problem of finding an optimal BDD is combinatorially hard and the size of the search space becomes intractable even for a small number of nodes. Sub-sampling is performed by randomly selecting a fraction r of samples from each dataset. For each of the selected datasets we sample 20 benchmarks for different values of the sampling ratio r.

Experiments were run on an Intel(R) i7 3370 3.40 GHz workstation with 4 cores and 16 GBytes of main memory, hosting a Ubuntu 14.04 LTS Linux distribution. We adopted a timeout of 3 hours and a memory limit of 4 GB for each run.

For each benchmark, we split the dataset into a training set and a test with a $80\%/20\%$ ratio. The first columns report the name, number of features (K), sampling ratio (r) and the number of samples in the sub-sampled benchmarks (M). The following columns provide the number of instances that were solved to optimality ($\#opt$), the average optimum BDD size (N_{avg}), the average total and SAT time (in seconds) and the average percentage accuracy. Results for each dataset and sampling ratio r are averaged over 20 runs, one for each of the randomly sampled benchmark. Averages are computed on the number of benchmarks for which we proved the optimum. Accuracy is computed as the percentage of samples in the test set that are correctly classified by the optimal BDD.

We have also run our heuristic method on the same set of benchmarks. Results are reported and compared in Table I.

				SAT					BDD		
$Name$	K	r	M	$\#opt$	N_{avg}	TOT_t	SAT_t	Acc_{avg}	N_{avg}	TOT_t	Acc_{avg}
A	530	0.05	5	20	3.1	743.5	13.1	80%	3.3	3.5	67%
		0.1	10	20	3.5	1692.9	18.3	85%	5.4	6.8	70%
		0.2	21	20	4.4	804.2	50.5	83%	11.2	14.1	71%
BC	41	0.05	13	20	4.35	1.40	0.1	69%	7.2	1.7	64%
		0.1	26	20	6.5	23.7	16.1	64%	13.9	3.2	61%
		0.2	53	7	8.8	3599.3	3562.1	70%	25.5	4.1	62%
Co	415	0.05	17	20	4.4	1798.6	17.5	64%	8.5	3.8	71%
		0.1	35	17	5.7	2953.7	196.1	76%	17.1	6.5	65%
		0.2	71	1	6.0	2158.8	281.9	66%	34.2	30.1	66%
Cl	395	0.05	15	20	4.1	314.5	7.6	77%	8.5	3.4	73%
		0.1	30	18	8.9	744.7	401.9	70%	15.7	10.9	59%

TABLE I: Comparison between SAT-based and heuristic approaches. Shortened benchmarks name are: [A] *appendicitis*, [BC] *breast-cancer*, [Co] *colic*, [Cl] *cleve*.

While data show that the SAT-based approach is poorly scalable, they provide a lower bound for BDD sizes. Though sizes computed by the heuristic approach are not far from the exact ones, they are in the range from ×1 to nearly ×5 the optimal value.

			COV			COF		
$Name$	K	M	N_{avg}	TOT_t	Acc_{avg}	N_{avg}	TOT_t	Acc_{avg}
chess	38	3196	17365	22	97%	470	22	89%
coil2000	654	1023	13284	1480	92%	187	378	89%
dis	1105	510	50	317	95%	10	582	95%
german	1073	511	93005	513	70%	219	402	65%
hypothyroid	1185	511	-	-	-	249	350	60%
mushroom	112	8204	101	73	100%	23	280	100%

TABLE II: Results on larger instances solved heuristically.

Table II focuses on the heuristic approach, by comparing the purely "sequential cover" approach (COV, implemented with covering functions available in the CUDD BDD package) to the one(s) based on cofactor (COF, interleaved variant). Experiments show that, though not in a purely uniform way, cofactoring reduces size w.r.t. COV. Accuracy values show slight changes, with a great variance among different benchmarks, intuitively related to the relevance of the chosen dataset.

VI. Conclusions

We have addressed the problem of deriving BDDs consistent with a given dataset. We have developed a SAT-based approach, to obtain a minimal BDD, as well as a heuristic one. The techniques can be used in different application domains. We have experimentally observed that the SAT-based approach, though with poor scalability, is able to compute a lower bound and we have shown that the more scalable heuristic approach is not far from the optimal value. Experimental data, though preliminary, also show that classification accuracy is more related to the quality of the data set than to the size minimization effort.

References

[1] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Inf. Process. Lett.*, vol. 5, no. 1, pp. 15–17, 1976.

[2] T. Hancock, T. Jiang, M. Li, and J. Tromp, "Lower bounds on learning decision lists and trees," *Inf. Comput.*, vol. 126, no. 2, p. 114–122, May 1996.

[3] L. Rokach and O. Maimon, *Data Mining With Decision Trees: Theory and Applications*, 2nd ed. USA: World Scientific Publishing Co., Inc., 2014.

[4] C. Bessiere, E. Hebrard, and B. O'Sullivan, "Minimising decision tree size as combinatorial optimisation," in *Principles and Practice of Constraint Programming - CP 2009*, I. P. Gent, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 173–187.

[5] N. Narodytska, A. Ignatiev, F. Pereira, and J. Marques-Silva, "Learning optimal decision trees with sat," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 1362–1368.

[6] F. Avellaneda, "Efficient inference of optimal decision trees," in *AAAI*, 2020, pp. 3195–3202.

[7] N. Narodytska, L. Ryzhyk, I. Ganichev, and S. Sevinc, "Bdd-based algorithms for packet classification," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 64–68.

[8] C. Y. Lee, "Representation of switching circuits by binary-decision programs," *The Bell System Technical Journal*, vol. 38, no. 4, pp. 985–999, 1959.

[9] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, 1978.

[10] M. Janota and A. Morgado, "Sat-based encodings for optimal decision trees with explicit paths," in *SAT*, 2020, pp. 501–518.

[11] A. Ignatiev, A. Morgado, and J. Marques-Silva, "PySAT: A Python toolkit for prototyping with SAT oracles," in *SAT*, 2018, pp. 428–437.

[12] G. Cabodi, S. Nocco, and S. Quer, "Benchmarking a model checker for algorithmic improvements and tuning for performance," *Formal Methods in System Design*, vol. 39, no. 2, pp. 205–227, 2011.

[13] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "Pmlb: a large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, no. 1, p. 36, Dec 2017.