Generative Model for Decision Trees

Riccardo Guidotti[1,2], **Anna Monreale**[1], **Mattia Setzu**[1], **Giulia Volpi**[1]

[1] University of Pisa, Pisa, Italy
[2] ISTI-CNR, Pisa, Italy
riccardo.guidotti@unipi.it, anna.monreale@unipi.it, mattia.setzu@unipi.it, giuliavolpi25.93@gmail.com

Abstract

Decision trees are among the most popular supervised models due to their interpretability and knowledge representation resembling human reasoning. Commonly-used decision tree induction algorithms are based on greedy top-down strategies. Although these approaches are known to be an efficient heuristic, the resulting trees are only locally optimal and tend to have overly complex structures. On the other hand, optimal decision tree algorithms attempt to create an entire decision tree at once to achieve global optimality. We place our proposal between these approaches by designing a generative model for decision trees. Our method first learns a latent decision tree space through a variational architecture using pre-trained decision tree models. Then, it adopts a genetic procedure to explore such latent space to find a compact decision tree with good predictive performance. We compare our proposal against classical tree induction methods, optimal approaches, and ensemble models. The results show that our proposal can generate accurate and shallow, i.e., interpretable, decision trees.

1 Introduction

Machine Learning (ML) techniques are currently employed in AI systems in high-stakes decision fields. The most effective ML predictors are considered "black-box" models (Guidotti et al. 2019b; Pasquale 2015) due to their complexity, which renders the decision process uninterpretable (Li et al. 2022; Miller 2019). However, interpretability is fundamental for predictive models adopted in sensitive domains (Freitas 2013; Mehrabi et al. 2022). Hence, there has recently been a flourishing of proposals for both post-hoc (Li et al. 2022; Guidotti et al. 2019b) and by-design explainability of AI models (Rudin 2019). Our proposal follows the latter direction by focusing on decision trees (DTs) (Breiman et al. 2017) that have a structure directly providing the learned decision logic in human-comprehensible terms (Craven et al. 1995; Guidotti et al. 2019a).

Finding optimal decision trees is an NP-hard problem (Hyafil et al. 1976) with prohibitive computational requirements that optimal induction algorithms can not really avoid (Bertsimas and Dunn 2017; Demirovic et al. 2022; Verwer et al. 2019; Khan et al. 2020; Blanquero et al. 2021; Mazumder et al. 2022). Instead, trees are most often learned through greedy sub-optimal induction algorithms, which yield decent performance at a fraction of the cost. Sub-optimal trees are also prone to overfitting, which pruning techniques attempt to minimize.

In between optimal and greedy sub-optimal trees, we find non-greedy sub-optimal trees such as *evolutionary* (Son 1998) and *genetic* (Kretowski 2004) trees, which provide a middle ground in terms of performance and complexity (Barros et al. 2012). Genetic induction algorithms encode trees in predefined representations, which are in turn optimized. Notably, since the encoding is predefined, the algorithm is limited by the quality of such representation, which can lead to unstable (Kretowski et al. 2006; Basgalupp et al. 2014) and sub-optimal solutions. To offset the performance gap, sub-optimal trees are combined into ensembles either through boosting (Chen et al. 2016) or bagging (Breiman 2001), which sacrifice interpretability for the sake of performance.

Our objective is to design a method to learn interpretable trees with high predictive performance and low complexity, overcoming the current limitations of state-of-the-art proposals. Therefore, we propose GENTREE, a generative model for decision trees able to induce sub-optimal and shallow decision trees. Notably, and unlike state-of-the-art induction algorithms, GENTREE is comprised of a *representation* model, which learns a subsymbolic latent space of trees, and an *optimization* model, which samples and optimizes trees from said latent space. Specifically, GENTREE implements the two with a Variational Autoencoder (VAE) (Kingma and Welling 2014) and a genetic algorithm (Eiben and Smith 2003), respectively. To the best of our knowledge, our proposal is the first tree induction algorithm which jointly exploits generative models and evolutionary algorithms to learn decision trees. We leverage the strong representation capabilities of VAEs, and the flexibility of evolutionary algorithms to *encode* and *search* trees with high performance and low complexity, respectively. We validated GENTREE on a large set of datasets for the classification problem, on which it performs favorably compared to classical decision trees and optimal trees, and on par w.r.t. trees ensembles.

2 Related Work

Decision trees are a widely used approach in supervised learning. Most induction algorithms recursively induce splits by locally optimizing a predefined split function. Several studies

point out that this approach is prone to overfitting (Hawkins 2004), attribute selection bias (Hothorn et al. 2006), and instability (Strobl et al. 2009). Alternatives such as ensembles of trees (Breiman 2001) have been proposed to overcome these problems (Seni et al. 2010; Hastie et al. 2009).

We loosely base our proposal on Meta-Heuristic (MH)-based approaches, which often induce highly accurate DTs (Rivera-López et al. 2022). Among them, those based on Evolutionary (EA) and Genetic (GA) Algorithms, and Swarm Intelligence (SI) stand out for their efficacy. These algorithms define an initial pool of *candidate trees*, which are encoded with a *predefined representation*, and then iteratively combined and validated through *variation operators* and a *fitness function*, respectively. EA and SI induction algorithms tend to employ different representations and variation operators. Representation typically encode candidate trees in vectors through appropriate conversion schemas (Wang et al. 2001). EA algorithms encode trees in *single* vectors where the split function and its parameters, i.e., split feature and threshold, are explicitly encoded (Kennedy et al. 1997). Others may instead encode the same information in a matrix (Vandewiele et al. 2016). Growing in encoding complexity, SI algorithms instead may encode trees with *pairs* of vectors, each vector encoding different characteristics of each node. As per variation operators, SI algorithms enjoy a large plethora of candidate optimizers: ant colony (Bursa et al. 2008), particle swarm, and bat swarm (Bida and Aouat 2021), each directly inspired by typical natural or animal ecosystems.

Our proposal is inspired by GENESIM (Vandewiele et al. 2016), an evolutionary DT induction algorithm that induces trees by evolving an initial set of trees, rather than inducing directly from data. However, unlike GENESIM and all other induction algorithms, we *learn* tree representations within a latent space, and then leverage GA to search for the best trees in said latent space. To the best of our knowledge, no other works jointly employ *learned* tree representations and optimization algorithms for decision tree induction.

3 Background

To keep our paper self-contained, we report here a brief overview of concepts necessary to comprehend our proposal.

Decision Trees. A Decision Tree (DT) is an interpretable predictive model (Guidotti et al. 2019b; Freitas 2013) representing its decisions through a structure composed of nodes and branches (Breiman et al. 2017; Tan, Steinbach, and Kumar 2005). DTs route instances within their structure, each node testing a split condition and routing instances towards its children, all the way down to the leaf nodes. Each instance thus traces a path inside the tree, effectively providing a *decision rule* describing the decision process of the tree on said instance. DTs are typically evaluated w.r.t. *accuracy* and *complexity* (Rokach and Maimon 2005), typically calculated as total number of nodes and leaves, tree depth, and number of attributes used. The simpler the tree, the more concise and interpretable the decision rules (Domingos 1999b; Endou et al. 2002; Cherkauer et al. 1996).

Split conditions, and thus trees, can be univariate (axis-parallel) or multivariate (oblique): the former operates on a single attribute, while the latter on multiple attributes. Multivariate trees generally perform better and are smaller than univariate trees when the training distribution is complex (Carreira-Perpiñán et al. 2018). However, axis-parallel DTs are much easier to interpret (Brodley and Utgoff 1995). Tree induction algorithms typically implement a top-down greedy search through the space of possible splits. CART (Breiman et al. 2017), ID3 (Quinlan 1986), and its successor C4.5 (Quinlan 1993) are the approaches that most exemplify this induction strategy. These recursive partitioning phase is usually followed by a pruning phase aimed to reduce complexity and overfit of the tree (Kotsiantis 2013).

Variational Autoencoders. Generative models have gained increasing interest due to their success in generating and representing data (Oussidi et al. 2018; Bengio et al. 2013). Among them, variational autoencoders (VAEs) (Kingma and Welling 2019, 2014) have proven particularly successful in a plethora of complex domains. VAEs model the data-generation process as a probability density $\Pr[X|Z,\theta]$ conditioned on a latent distribution Z and parameters θ. Training such a model requires sampling over an intractably large dense sampling space, which renders optimization intractable. Thus, VAEs employ a two-tier architecture comprised of an *encoder* Q_ϕ, which models $\Pr[Z|X]$ and renders the optimization tractable by drastically reducing the sampling space for Z, and a *decoder* P_ψ, which models $\Pr[X|Z]$. The two components are jointly optimized through stochastic gradient descent by optimizing the ELBO loss (Yang 2017):

$$\mathbb{E}_{Q_\phi(X|Z)}[\log P(X|Z)] - \mathcal{D}_{KL}(Q_\phi(Z \mid X) \mid\mid P_\psi(Z)),$$

where the two terms work in concert to optimize data likelihood. Encoder and decoder are typically neural networks, which makes VAEs extremely flexible and powerful. VAEs have most widely been used to model complex data, including images (Razavi et al. 2019), text (Wang et al. 2019), and even discrete data such as logic programs (Misino et al. 2023), yet, to the best of our knowledge, this is their first application on interpretable models, and specifically on decision trees.

Evolutionary Algorithms. Evolutionary Algorithms (EAs) are optimization algorithms inspired by Darwinian evolution (Darwin 1909). Starting from an initial set of candidate solutions, encoded as *chromosomes* through a suitable *encoding function* π, EAs iteratively optimize them through two variation operators, *crossover* and *mutation*. The former combines existing chromosomes into novel ones, while the latter randomly perturbs them. Chromosomes are evaluated according to their *fitness*, which rewards better solutions by driving further crossovers, and penalizes worse solutions by introducing random mutations. A *stopping criterion* halts the algorithm after a number of iterations, and the fittest chromosome is returned (Eiben and Smith 2003). Notably, EAs allow us to achieve a *set* of solutions, rather than a single one. Due to their flexibility, several induction algorithms already employ EAs (Koza 1990; Vandewiele et al. 2016; Jankowski et al. 2014; Zhao 2007; Carvalho et al. 2000; Turney 1995; Fu et al. 2003; Rivera-López et al. 2018) some with vectorized (Kennedy et al. 1997; Cha and Tappert

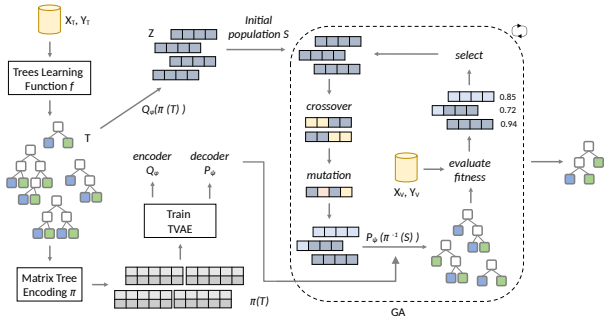

Figure 1: GENTREE workflow: *i)* learns a set T of decision trees, *ii)* trains a Tree-VAE to learn a latent space of trees, and finally, *iii)* searches for trees in the learned latent space.

2009), others with structured chromosomes (Papagelis et al. 2000; Podforelec et al. 1998; Kalles et al. 2010).

4 Generative Tree Model

In this section, we present GENTREE, a GENerative model for decision TREEs as an alternative procedure for inducing DTs. GENTREE first learns a latent decision tree distribution through a VAE by leveraging pre-trained DTs, and then adopts a GA to explore said latent space and builds a shallow tree with good predictive performance. The pseudo-code of GENTREE is reported in Algorithm 1 and illustrated in Figure 1. GENTREE takes as input a dataset X, Y where $X = \{x_1, \ldots, x_n\} \in \mathbb{R}^{n \times m}$ is a set of n records described by m attributes (features), and $Y = \{y_1, \ldots, y_n\} \in \mathbb{R}^n$ is the set of the target variable.[1] When dealing with classification, $y_i \in [0, \ldots, l-1]$ and l is the number of the classes, while when dealing with regression, $y_i \in \mathbb{R}$. The output of GENTREE is a shallow and accurate decision tree t that minimizes the predictive error. We present GENTREE for classification, however, it can be also adopted also for regression.

GENTREE starts by inducing a set $T = \{t_1, \ldots, t_n\}$ of n decision trees (line 1) with function f implemented either by leveraging existing suboptimal tree induction algorithms such as CART (Breiman et al. 2017) or C4.5 (Quinlan 1993) repeated n times, each inducted independently with different hyperparameters, or by directly learning a tree ensemble, e.g., a Random Forest (RF) (Breiman 2001). Indeed, the trees in T can rely on different features, have different structures and different depth. Then, we encode the trees in a fixed-length matrix representation suitable for VAEs (line 2) through an invertible encoding function π, and train the VAE (Q_ϕ, P_ψ) (line 3). Chaining π and Q_ϕ we are able to encode a decision tree into a continuous vector representation, while chaining P_ψ and π^{-1} we can map back a continuous tree representation to an actual decision tree. Specifically, we leverage a TVAE architecture (Xu et al. 2019). Using the encoding $Q_\phi(\pi(T))$ of the training trees T, GENTREE initializes the set of candidate solutions (line 5), and proceeds to iteratively optimize it (lines 6-11). Once the stopping condition is met, GENTREE selects (line 12) and returns (line 13) the best solution. Note that GENTREE provides flexible tree representations in the latent space alongside an optimization algorithm as jointly trained decoupled components. As such, unlike state-of-the-art induction algorithms, GENTREE is particularly modular and agnostic to its components. The VAE component can be replaced by any model able to provide latent representations of trees, and likewise the tree optimization algorithm can be replaced by other optimization algorithms such as Monte Carlo and simulated annealing. In the following, we provide details for the matrix coding function π, the $TVAE$ architecture, and the steps of the EA.

[1] In the algorithm, X, Y are already split into a training set (X_T, Y_T) for tree induction, and a set X_V, Y_V for fitness evaluation.

Algorithm 1: GENTREE(X_T, Y_T, X_V, Y_V)

Input : (X_T, Y_T) - training dataset, (X_V, Y_V) - training dataset comprised of decision trees
Params : f - trees induction algorithm, λ_T, λ_V - trees and VAE hyperparameters, g, n_S - number of generations and population size, p_c, p_m - probability of crossover and mutation, *fitness* - fitness function
Output : t^* - decision tree

1 $T \leftarrow f(X_T, Y_T, \lambda_T)$; // induce trees
2 $M_T \leftarrow \pi(T)$; // encode the trees
3 $Q_\phi, P_\psi \leftarrow VAE(M_T, \lambda_V)$; // train the VAE
4 $Z_T \leftarrow Q_\phi(\pi(T))$; // latent trees encoding
5 $S \leftarrow chromosomes(Z_T, n_S)$; // initial n_S solutions
6 **for** $i \in [1, g]$ **do** // for each generation
7 $S \leftarrow crossover(S, p_c)$;
8 $S \leftarrow mutate(S, p_m)$;
9 $T \leftarrow \pi^{-1}(P_\psi(S))$; // decode trees
10 $F \leftarrow fitness(T, X_V, Y_V)$; // evaluate trees
11 $S \leftarrow filter(S, F, n_S)$; // best current solutions
12 $t^* \leftarrow arg\max_T F$; // select best tree
13 **return** t^*;

Matrix Representation of Trees. Both the representation and optimization components are strongly dependent on the data representation, the VAE for correctly encoding trees, and the EA for correctly manipulating them and evaluating them. To accommodate both components, in line with (Jankowski et al. 2014), we define an invertible encoding function $\pi : \mathcal{T} \rightarrow \mathbb{R}^{2 \times 2^{\bar{d}} - 1}$ able to map decision trees into $2 \times 2^{\bar{d}} - 1$ matrices. Thus, we encode a tree on a $\mathrm{R}^{2 \times 2^{\bar{d}} - 1}$ matrix, where the two rows hold information regarding the nature of the nodes. The first row holds the split feature of each node if an internal node, or -1 if a leaf, while the second row holds the split threshold for internal nodes, and the classification label for leaves. Tree navigation follows the conventional breadth-first practices, i.e., the i-th visited node is found in column i, and its children in columns $2i$ and $2i + 1$, respectively. Figure 2 shows an example of a matrix representation of a DT of depth 3. On the right, we can see the result of the expansion process applied to the left child of the root node. The decoding procedure π^{-1} generates a binary tree from the matrix and applies a pruning step to eliminate duplicate

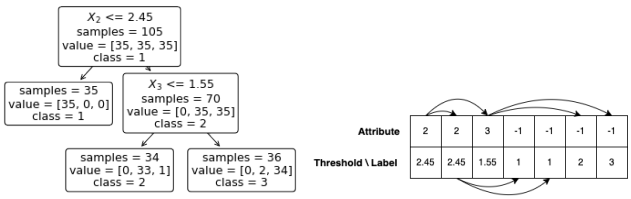

Figure 2: DT and corresponding matrix representation. The matrix holds node information (top row, splitting feature for internal nodes, -1 for others), and node parameters (bottom rowsplit threshold for internal nodes, class for leaves).

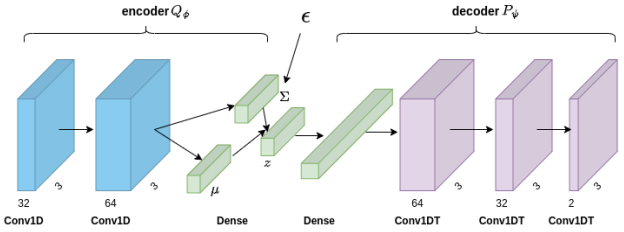

Figure 3: TVAE architecture. The noise ϵ renders sampling, and thus the whole architecture, fully differentiable.

leaves, i.e., children of a node classifying a sample with the same class label. Incomplete trees and trees with depth lower than a maximum predefined depth $\overline{d}$ are appropriately padded, thus avoiding data sparsity, which can negatively impact VAE training (Zhao et al. 2020). Nodes at the penultimate layer are replicated appropriately to fill the whole matrix.

Tree Variational Autoencoder. The representation component is implement with a Tree-VAE (TVAE) (Sohn, Lee, and Yan 2015) – see Figure 3. The encoder Q_ϕ is a convolutional network with two convolutional layers[2] followed by a linear one, which maps into the latent representation of dimensionality $k = \lceil \log_2(d) \rceil$. The decoder network P_ψ mirrors the encoder. The TVAE is trained on a set of n decision trees (line 3, Algorithm 1) for g epochs[3]. We highlight that the TVAE is only tasked with learning proper tree representation, while the optimization of such representation is delegated to the optimization module. However, even though the TVAE is trained on DTs limited in performance, this would not prevent GENTREE from finding tree representations which are suboptimal oras long as the optimal tree is not extremely different from the training trees, even optimal.

Genetic Tree Latent Space Exploration. The optimization component, implemented as GA, allows us to induce a decision tree by optimizing a set of initial candidate trees. Thanks to the mapping in a latent space provided by the VAE, the GA can operate in a more favorable chromosome space. Specifically, the latent mapping *i)* reduces the dimensionality of the space of solutions, thus reducing the search space and cost of the optimization algorithm; *ii)* transforms the discrete space of decision trees in a continuous one, which is more amenable to optimization and *iii)* has higher expressive power (Bengio et al. 2013).

We employ a standard Genetic optimization Algorithm: given an initial set of candidate solutions (line 1 of Algorithm 1) appropriately encoded as chromosomes (lines 2–5), we iteratively employ crossover (line 7) and mutation (line 8) operations, evaluate the fitness of the resulting chromosomes (line 10), and filter out the worst current solutions (line 11). Fitness is evaluated on the actual, rather than encoded, trees, which are obtained by first decoding chromosomes through the decoder P_ψ, and then decoding the matrix representation through π^{-1}. Notably, by explicitly decoding the decision trees, we can leverage already-existing metrics for tree validation. Here, we define fitness to reward accurate as well as shallow, and thus more comprehensible (Endou et al. 2002; Cherkauer et al. 1996), and simple (Domingos 1999a) trees. Inspired by (Tan, Steinbach, and Kumar 2005), we propose two fitness functions:

$$1 - acc(Y, Y^*) + \omega \frac{\#leaves}{\mid X_V \mid} + \Lambda(t) \qquad (1)$$

$$1 - (1 - \omega) acc(Y, Y^*) - \omega \frac{1}{\#nodes + 1} + \Lambda(t) \qquad (2)$$

where $acc(Y, Y^*)$ is the accuracy of the DT on X, $\omega \in [0, 1]$ is a weight aimed at balancing complexity over the accuracy. Finally, $\Lambda : \mathcal{T} \rightarrow \{0, +\infty\}$ is a function that checks the validity of a tree t: if t is a valid tree it returns 0, otherwise it returns $+\infty$. The GA aims to minimize the fitness function, i.e., to minimize the error rate and the tree complexity simultaneously. Thus, chromosomes leading to an invalid tree are automatically discarded[4].

5 Experiments

The experiments for validating are GENTREE presented here. First, we illustrate the experimental setting. Then, we analyze the sensitivity of the hyperparameters of GENTREE. After that, we compare the DT induced by GENTREE against some competitors. Finally, we study the tree latent space learned.

Experimental Setting. We experimented on 18 datasets from UCI Machine Learning Repository.[5] We experimented on datasets with only continuous attributes, leaving as future work the study of the effect of categorical attributes on the encodings. Details[6] are reported in Table 1. Each dataset was randomly split as 80%-10%-10% into a train set (X_T, Y_T), used to induce a set of trees T; validation (X_V, Y_V), employed for calculating the fitness function; and test, used to measure the performance and complexity of GENTREE and it competitors, including running time[7]

[2]Layers implement a 1-D convolution with 32 and 64 channels, respectively, kernel size of 3 and stride of 1.

[3]Up to g epochs, due to early stopping.

[4]In practice, this happens a negligible amount of times.

[5]https://archive.ics.uci.edu/ml/index.php.

[6]Some dataset names are trimmed due to space: `aus` for `australian`, `bnk` for `banknote`, `brst` for `breast`, `dbn` stands for `drybean`, `iso` for `isolet`, and `vlc` for `vehicle`.

[7]Run on Windows 10, 16GB RAM, 1.80GHz Intel Core i7.

Dataset	n	m	$\mid \mathcal{Y} \mid$	% majority	% minority	$\overline{d}$
aus	690	38	2	55.51	44.49	4
bnk	45211	48	2	88.30	11.70	4
bnote	1372	4	2	55.54	44.46	6
brst	699	9	2	65.52	34.48	6
cars	1728	6	4	70.02	3.76	8
dbn	13611	16	7	26.05	3.84	7
ecoli	327	6	5	43.73	6.12	4
glass	214	9	6	35.51	4.21	4
heart	270	20	2	55.56	44.44	4
iris	150	4	3	33.33	33.33	4
iso	7797	617	26	3.85	3.82	10
led7	2563	7	8	13.30	10.53	7
lymph	142	47	2	57.04	42.96	4
pima	768	8	2	65.10	34.90	4
sonar	208	60	2	53.37	46.63	7
vlc	846	18	4	25.77	23.52	10
wine	6497	11	7	43.65	0.08	9
yeast	1484	8	10	31.20	0.34	6

Table 1: Datasets size (n), dimensionality (m), class number ($\mid \mathcal{Y} \mid$), and majority and minority class percentages.

	fitness (1)		*fitness* (2)	
	Accuracy	**Complexity**	**Accuracy**	**Complexity**
CART	$.832 \pm .12$	13.0 ± 10.46	$0.638 \pm .16$	2.0 ± 1.15
RF	$.699 \pm .20$	11.5 ± 10.75	$0.653 \pm .28$	5.0 ± 1.63

Table 2: Impact of different learning and fitness functions.

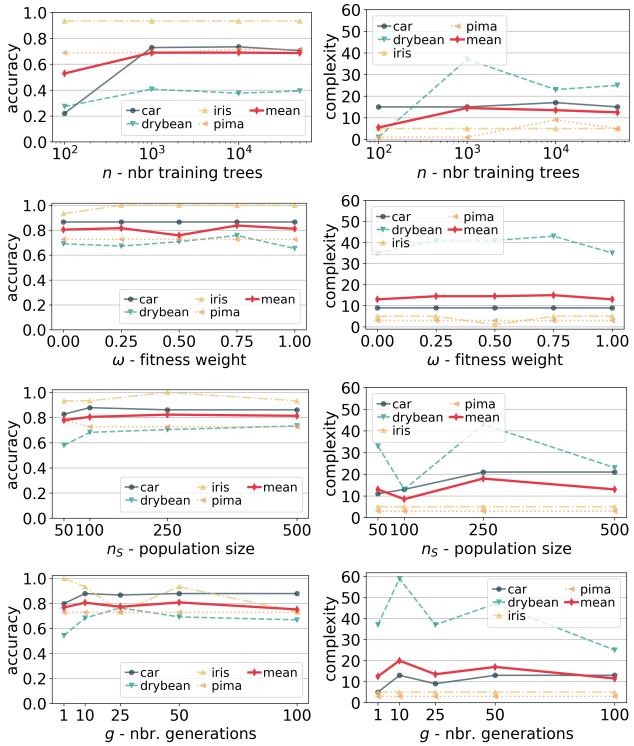

Figure 4: Impact of number of trees n, the complexity weight ω, genetic population size n_S, and number of iterations g.

(in seconds). We implemented GENTREE in Python[8]. We experimented with the following hyperparameters: tree induction algorithms f (CART and Random Forest (RF)), genetic population size ($\{50, 100, 250, 500\}$), complexity weight ω ($\{0.0, 0.25, 0.5, 0.75, 1.0\}$), number of generations g ($\{1, 10, 25, 50, 100\}$), and *fitness* function ((1) and (2) as defined in Equation 1). Probability of crossover and mutation are set as specified in (Alan de Jong 1975). The maximum tree depth $\overline{d}$ is specifically selected for each dataset by training a DT using CART with growing maximum depth (2 to 100), and then selecting the maximum depth $\overline{d}$ yielding the tree with maximum accuracy.

Sensitivity Analysis. We analyze here the impact of some hyperparameters of GENTREE on `cars`, `dbn`, `iris`, and `pima`. In Table 2 we compare alternative trees learning functions f and fitness functions *fitness*. Fitness function (1) achieves high accuracy and high complexity (while maintaining a small variability), while with (2) GENTREE prunes the trees too much and obtains very small but inaccurate trees. Thus, in the following, we consider GENTREE models trained on CART trees and trained with fitness (1). After that, in Figure 4 we observe how the accuracy and the complexity change when varying *(i)* the number of trees, *(ii)* the fitness weight ω, *(iii)* the size of the population for the genetic algorithm, and *(iv)* the number g of iteration of the genetic algorithm. We can notice that increasing the number of trees increases the accuracy only up to 10000. Also, the complexity for less than 10000 trees is sensibly higher[9]. We observe a similar effect varying the size of the genetic population. We did not observe any improvement using more than 100 DTs, which also yields the lowest complexity. Regarding ω, GENTREE appears to be be relatively insensitive to it, achieving stable results, with a peak around $\omega = 0.75$. Finally, it seems that it is better to keep the number of genetic iterations g not higher than 10 to avoid a decrease in the performance.

Tree Induction Methods Comparison. We compare GENTREE (GT) against *(i)* a CART tree (DT) (Breiman et al. 2017) as implemented by `sklearn`, *(ii)* Optimal Decision Trees[10] (ODT) (Bertsimas and Dunn 2017), *(iii)* GENESIM (GS) w.r.t. the datasets analyzed in (Vandewiele et al. 2016), and *(iv)* Random Forest (RF). For GENESIM (GS) (Vandewiele et al. 2016), the ensemble to be transformed into a single decision tree was constructed by applying several induction algorithms (C4.5, CART, QUEST (Loh 2008), and GUIDE (Loh 2009)) combined with bagging and boosting. For DT, ODT and RF we adopt the same maximum depth $\overline{d}$ value adopted for GT, while for the other hyperparameters we use default

[8]https://github.com/msetzu/GenTree.

[9]Since the TVAE learns a latent space whose complexity makes it more or less amenable to optimization, i.e., the more entangled the learned space, the more complex it is to optimize over such space, we have opted for relatively simple TVAEs.

[10]https://docs.interpretable.ai/stable/OptimalTrees/

	Accuracy ↑					Complexity ↓				
	DT	GT	GS	ODT	RF	DT	GT	GS	ODT	RF
aus	.829	*.855*	*.855*	*.855*	**.858**	29.0	**3.0**	23.8	**3.0**	26.7
bank	.825	*.891*	-	**.894**	.881	31.0	*11.4*	-	**11.0**	30.4
bnk	.916	.915	-	*.978*	**.996**	43.0	**11.0**	-	*19.0*	40.8
brst	.864	.914	*.950*	.929	**.957**	41.0	**3.4**	18.5	*7.0*	42.1
car	.866	.875	-	*.913*	**.954**	79.0	**19.6**	-	*49.0*	147.0
dnb	.817	.665	-	**.902**	*.897*	183.0	**73.6**	-	*75.0*	169.7
ecoli	.850	.760	*.853*	**.940**	*.916*	25.0	**3.0**	*19.1*	15.0	26.4
glass	.716	*.777*	.670	.682	**.855**	23.0	**5.8**	29.7	*7.0*	23.8
heart	.764	.784	*.798*	.780	**.920**	29.0	*6.2*	17.4	**3.0**	26.8
iris	.913	**.967**	*.946*	.933	.933	13.0	**5.0**	*5.9*	7.0	12.0
iso	.730	.779	-	*.822*	**.933**	*526.2*	815.2	-	**163.0**	597.4
led7	.733	.753	*.793*	.795	**.804**	217.0	**51.2**	92.0	*57.0*	195.8
lymph	.797	*.800*	.787	*.800*	**.920**	25.0	**5.0**	*14.8*	**5.0**	21.5
pima	.644	.726	*.727*	.766	**.797**	31.0	*3.6*	45.2	**3.0**	28.5
sonar	.711	.695	-	*.762*	**.900**	39.0	**23.0**	-	*31.0*	40.0
VCL	.649	.678	.683	**.776**	*.764*	217.8	**41.8**	83.2	*49.0*	194.7
wine	.482	.481	**.913**	.503	*.595*	660.2	39.6	**8.0**	*11.0*	528.6
yeast	.509	.519	-	*.530*	**.572**	95.0	**28.4**	-	*35.0*	86.3
avg	.753	.779	*.813*	.808	**.858**	128.1	62.7	*32.5*	**30.5**	124.3
rank	2.10	1.78	-	*1.43*	**0.99**	2.10	*1.10*	-	**1.06**	1.86

Table 3: Methods accuracy and complexity. Average score and average rank position are reported on the bottom. In bold the best performer. In italics the best performer runner-up.

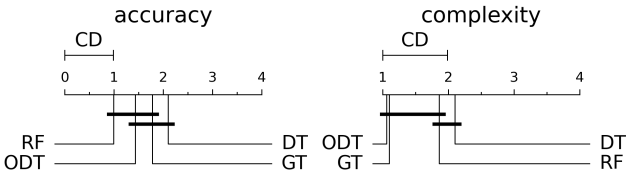

Figure 5: Critical Difference plots with Nemenyi at 90% confidence level for accuracy and complexity.

values. Competitors use the concatenation of (X_T, Y_T) and (X_V, Y_V) as development set to induce trees. To guarantee a statistically valid evaluation of the performance, as proposed in (Rajkomar et al. 2018), we bootstrapped each test set 10 times, and we report the mean values obtained by the various methods over these runs. We do not compare against other methods because their implementation/experiments on open source datasets are not available, and because our objective is to show that GENTREE performance are bounded by traditional DT and ODT/RF.

Table 3 reports the accuracy and complexity for each dataset and the average score and rank (not considering GS). With respect to complexity, RF is nearly always the best performer, followed by ODT. The second best performer is GT, which is (nearly) always better than DT. At the same time, GT is the second-best performer in terms of complexity, returning trees with structures similar to those returned by ODT. GT always returns trees less complex than GS while having comparable accuracy. The non-parametric Friedman test that compares the average ranks of tree induction methods over multiple datasets w.r.t. accuracy and complexity guarantees that these results are statistically significant, i.e.,

	Tree induction				GENTREE		
	DT	GT	ODT	RF	TVAE	$Q_\phi \circ \pi$	GA
μ	0.188	290.12	638.43	0.81	221.12	6.55	62.18
σ	0.739	468.06	2600.87	2.02	327.12	7.29	113.22

Table 4: Mean (μ) and std. dev. (σ) of runtime over different datasets (left). Details of the runtime of GENTREE (right).

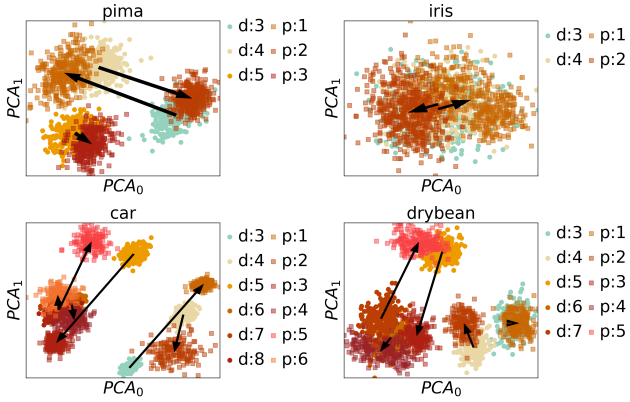

Figure 6: GENTREE latent tree spaces shown with two principal components for 1000 decision trees and their pruned counterpart. Depths (after ":") are colored in different ways.

the null hypothesis that all methods are equivalent is rejected ($p\text{-}value < .0001$). Also, the comparison of the ranks against each other is represented in Figure 5 with critical difference diagrams (Demsar 2006). Two methods are tied if the null hypothesis that their performance is the same cannot be rejected using the Nemenyi test at $\alpha = .1$. We can notice that GT is always tied with ODT, and in terms of complexity performs significantly bettern than DT. Table 4 reports on the left the total runtime, while on the right, the details for different aspects of GENTREE. We notice that GT and ODT are markedly slower than DT and RF, the former $\times 2.2$ than the latter. GENTREE displays a TVAE-induced bottleneck, whose impact can be easily reduced with GPU training.

Latent Tree Space Inspection. To understand the expressive power and the properties held by the newly defined latent tree space, we study here the latent tree space learned by GENTREE for `car`, `dbn`, `iris` and `pima`. We wish to understand if similarity in this space is invariant to tree transformations, i.e., if tree transformations, e.g., pruning, affect the similarity of trees in latent space. In Figure 6 we show the scatter plots of the latent tree space representation of 1000 CART decision trees *before* (DTd) and *after* two-level ($pDTd$) pruning. Different colors highlight different depths, and black arrows show how trees move in the space when pruned. We notice two different behaviors. For `pima`, it seems that the PCA representation with two components fits well to represent the latent tree space. DT with depth 5 ($DTd = 5$) when pruned moves closer towards the right, becoming pruned DT with depth 3 ($pDTd = 3$, bottom left of the plot). On the other hand, when pruning $DTd = 3$ and $DTd = 4$, we witness a

dataset	labeling	Z	Z'	$Z' - Z$
pima	clust	1.519 ± 0.54	1.519 ± 0.54	0.161 ± 0.11
	rand	2.976 ± 1.26	2.976 ± 1.26	0.754 ± 0.49
iris	clust	3.625 ± 1.28	3.625 ± 1.28	0.786 ± 0.43
	rand	4.713 ± 1.30	4.713 ± 1.30	0.977 ± 0.46
car	clust	1.635 ± 0.59	1.635 ± 0.59	0.127 ± 0.09
	rand	4.097 ± 1.53	4.097 ± 1.53	0.721 ± 0.43
dbn	clust	1.627 ± 0.57	1.627 ± 0.57	0.127 ± 0.08
	rand	3.977 ± 1.36	3.977 ± 1.36	0.901 ± 0.46

Table 5: SSE on groups of latent DTs Z, latent pDTs Z', and latent direction $Z' - Z$ with groups obtained *(i)* with *clus*tering on Z, or with *rand*om label assignment.

sort of exchange in position in the scatter plot. Thus, similar trees in the latent tree space are typically close to each other and clustered, but when they are pruned, we cannot guarantee that they move near to the original position. However, they massively move to another area remaining clustered. Similar effects are observed for car and dbn. On the other hand, w.r.t. iris, it seems that there is not a clear separation among trees, at least w.r.t. the PCA representation[11].

We further inspected the tree latent space with K-Means (Tan, Steinbach, and Kumar 2005) on the tree latent space of 1000 DT with maximum depth $\overline{d}$. By knee curve analysis, K was set to 10 for all datasets except for car, where it was set to 4. After having clustered the latent trees (DT) Z, we applied the same cluster labels *(i)* to the corresponding pruned DT (pDT) Z', *(ii)* to the directions in the latent space to pass from DT to pDT, i.e., the point-to-point difference $Z' - Z$. Then, we calculated the Sum of Squared Error (SSE) on these partitionings, and we compared it with the SSE of 100 random assignments still w.r.t. 10/4 clusters[12]. The results in Table 5 prove that the two different label assignment strategies are independent and the random ones never succeed in grouping closer elements of Z' and $Z' - Z$ w.r.t. the one based on the clustering on the latent space of DT (Nguyen et al. 2010). Hence, pruning (or growing) in the same way similar DTs in the latent space reflects in obtaining similar DTs along the same movements.

We visually inspect Cluster 8 of iris to better understand this phenomenon. The first row of Figure 7 reports a scatter plot on the cluster unveiling that, also for iris, at the cluster level, we observe a movement toward another latent tree area when pruning is applied. The bar plots show the average feature importance with deviations for the original decision trees (DT) and their pruned versions (pDT). We notice that when trees are pruned, the importance of the features focuses on a smaller set of attributes. In this example, for both clusters, the second most important attribute shows very high variability. The second row of Figure 7 reports two trees for each cluster, and the corresponding pruned version appears on the third row. All the trees are highlighted with bigger markers in the scatter plots. We notice that the two trees differ because the right one has all pure leaves and, therefore, an additional

[11]Comparable results are obtained using t-SNE.

[12]For Z and Z' the SSE was calculated with Euclidean distance, while for $Z' - Z$ with the Cosine distance.

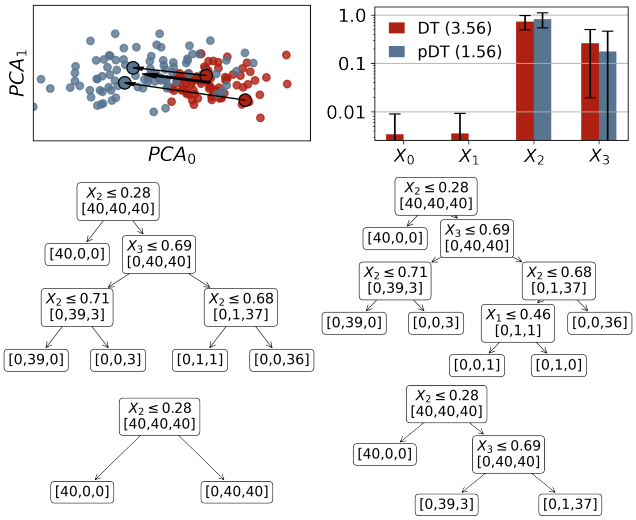

Figure 7: Visualizations of Cluster 8 for iris. First row: focused scatter plots on the clusters and average features importance with deviations. Second row: samples of original trees. Third row: corresponding pruned trees.

level. However, all the splits are the same. When these trees are pruned, we observe the same difference in depth and a similar direction in the latent tree space.

6 Conclusion

We have presented GENTREE, a decision tree induction algorithm powered by latent tree representation and genetic tree optimization. Experimental results show that the performance of the trees returned by GENTREE are better than those obtained by traditional trees and on par with those obtained by optimal decision trees, still guaranteeing a lower running time. An advantage of GENTREE over other algorithms based on genetic algorithms is that genetic operators can be applied without any modification since an individual is a real-valued vector. Besides, the inspection of the latent tree space learned by GENTREE shows interesting properties that can be exploited for future studies.

Several extensions and additional experiments can be mentioned as future works. First, GENTREE does not implement any local control on the trees during the training phase of the TVAE that allows to generate only DT also having a wrong or incoherent structure. Also, it does not control any global property of the space learned. Instead, the control of the correctness is delegated to the evaluation step of the GA. Thus, we would like to extend the TVAE such that it only generates correct DTs and such that the latent tree space might guarantee desired properties. Second, another study could focus on implementing alternative techniques to map DTs into real-value vectors and assess their impact on GENTREE. Finally, the framework introduced by GENTREE, which consists of using a pipeline of a representation model followed by an optimization component to extract an interpretable model for decision making, could be extended to different combinations of algorithms and models.

Acknowledgments

This work has been partially supported by the European Community H2020 programme under the funding schemes ERC-2018-ADG G.A. 834756 "XAI: Science and technology for the eXplanation of AI decision making" (https://xai-project.eu/), "INFRAIA-01-2018-2019 – Integrating Activities for Advanced Communities", G.A. 871042, "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics" (http://www.sobigdata.eu), G.A. 952026 HumanE AI Net (https://www.humane-ai.eu/), G.A. 952215 TAILOR (https://tailor-network.eu/), G.A. 101120763 TANGO, by the European Commission under the NextGeneration EU programme – National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) – Project: "SoBigData.it – Strengthening the Italian RI for Social Mining and Big Data Analytics" – Prot. IR0000013 – Avviso n. 3264 del 28/12/2021, and M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI", and by the Italian Project Fondo Italiano per la Scienza FIS00001966 MIMOSA. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or of the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

Alan de Jong, K. 1975. *Analysis of the behavior of genetic adaptive systems*. Ph.D. thesis, University of Michigan, USA.

Barros, R. C.; et al. 2012. A Survey of Evolutionary Algorithms for Decision-Tree Induction. *IEEE Trans. Syst. Man Cybern. Part C*, 42(3): 291–312.

Basgalupp, M. P.; et al. 2014. Evolving decision trees with beam search-based initialization and lexicographic multi-objective evaluation. *Inf. Sci.*, 258: 160–181.

Bengio, Y.; et al. 2013. Representation Learning: A Review and New Perspectives. *IEEE PAMI*, 35(8): 1798–1828.

Bertsimas, D.; and Dunn, J. 2017. Optimal classification trees. *MACH*, 106(7): 1039–1082.

Bida, I.; and Aouat, S. 2021. Swarm Intelligence-based Decision Trees Induction for Classification—A Brief Analysis. In *IHSH*, 165–170. IEEE.

Blanquero, R.; et al. 2021. Optimal randomized classification trees. *Comput. Oper. Res.*, 132: 105281.

Breiman, L. 2001. Random forests. *MACH*, 45(1): 5–32.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 2017. *Classification and regression trees*. Routledge.

Brodley, C. E.; and Utgoff, P. E. 1995. Multivariate Decision Trees. *MACH*, 19(1): 45–77.

Bursa, M.; et al. 2008. Automated Classification Tree Evolution Through Hybrid Metaheuristics. In *Innov. in Hybr. Intell. Sys.*, volume 44 of *Adv. in Soft Comp.*, 191–198. Springer.

Carreira-Perpiñán, M. Á.; et al. 2018. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *NeurIPS*, 1219–1229.

Carvalho, D. R.; et al. 2000. A hybrid decision tree/genetic algorithm for coping with the problem of small disjuncts in data mining. In *GECCO*, 1061–1068.

Cha, S.-H.; and Tappert, C. C. 2009. A genetic algorithm for constructing compact binary decision trees. *Journal of pattern recognition research*, 4(1): 1–13.

Chen, T.; et al. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD*, 785–794. ACM.

Cherkauer, K. J.; et al. 1996. Growing Simpler Decision Trees to Facilitate Knowledge Discovery. In *KDD*, 315–318. AAAI Press.

Craven, M. W.; et al. 1995. Extracting Tree-Structured Representations of Trained Networks. In *NIPS*, 24–30. MIT.

Darwin, C. 1909. *The origin of species*. PF Collier & Son.

Demirovic, E.; et al. 2022. MurTree: Optimal Decision Trees via Dynamic Programming and Search. *JMLR*, 23: 1–47.

Demsar, J. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, 7: 1–30.

Domingos, P. M. 1999a. MetaCost: A General Method for Making Classifiers Cost-Sensitive. In *KDD*, 155–164. ACM.

Domingos, P. M. 1999b. The Role of Occam's Razor in Knowledge Discovery. *DAMI*, 3(4): 409–425.

Eiben, A. E.; and Smith, J. E. 2003. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer.

Endou, T.; et al. 2002. Generation of comprehensible decision trees through evolution of training data. In *IEEE CEC*, 1221–1225. IEEE.

Freitas, A. A. 2013. Comprehensible classification models: a position paper. *SIGKDD Explor.*, 15(1): 1–10.

Fu, Z.; et al. 2003. A Genetic Algorithm-Based Approach for Building Accurate Decision Trees. *INFORMS*, 15(1): 3–22.

Guidotti, R.; et al. 2019a. Factual and Counterfactual Explanations for Black Box Decision Making. *IEEE Intell. Syst.*, 34(6): 14–23.

Guidotti, R.; et al. 2019b. A Survey of Methods for Explaining Black Box Models. *ACM CSUR*, 51(5): 93:1–93:42.

Hastie, T.; et al. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Statistics. Springer.

Hawkins, D. M. 2004. The Problem of Overfitting. *J. Chem. Inf. Model.*, 44(1): 1–12.

Hothorn, T.; et al. 2006. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3): 651–674.

Hyafil, L.; et al. 1976. Constructing Optimal Binary Decision Trees is NP-Complete. *Inf. Process. Lett.*, 5(1): 15–17.

Jankowski, D.; et al. 2014. Evolutionary Algorithm for Decision Tree Induction. In *CISIM*, volume 8838 of *LNCS*, 23–32. Springer.

Kalles, D.; et al. 2010. Lossless fitness inheritance in genetic algorithms for decision trees. *Soft Comput.*, 14(9): 973–993.

Kennedy, H.; et al. 1997. The Construction and Evaluation of Decision Trees: a Comparison of Evolutionary and Concept Learning Methods. In *Evolutionary Computing, AISB*, volume 1305 of *LNCS*, 147–162. Springer.

Khan, Z.; et al. 2020. Ensemble of optimal trees, random forest and random projection ensemble classification. *Adv. Data Anal. Classif.*, 14(1): 97–116.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Kingma, D. P.; and Welling, M. 2019. An Introduction to Variational Autoencoders. *FTML*, 12(4): 307–392.

Kotsiantis, S. B. 2013. Decision trees: a recent overview. *Artif. Intell. Rev.*, 39(4): 261–283.

Koza, J. R. 1990. Concept Formation and Decision Tree Induction Using the Genetic Programming Paradigm. In *PPSN*, volume 496 of *LNCS*, 124–128. Springer.

Kretowski, M. 2004. An Evolutionary Algorithm for Oblique Decision Tree Induction. In *ICAISC*, volume 3070 of *LNCS*, 432–437. Springer.

Kretowski, M.; et al. 2006. Mixed Decision Trees: An Evolutionary Approach. In *DaWaK*, volume 4081 of *LNCS*, 260–269. Springer.

Li, X.; et al. 2022. A Survey of Data-Driven and Knowledge-Aware eXplainable AI. *IEEE TKDE*, 34(1): 29–49.

Loh, W.-y. 2008. Classification and Regression Tree Methods. *Encyclopedia of statistics in quality and reliability*.

Loh, W.-Y. 2009. Improving the precision of classification trees. *The Annals of Applied Statistics*, 1710–1737.

Mazumder, R.; et al. 2022. Quant-BnB: A Scalable Branch-and-Bound Method for Optimal Decision Trees. In *ICML*, volume 162 of *PMLR*, 55–77. PMLR.

Mehrabi, N.; et al. 2022. A Survey on Bias and Fairness in Machine Learning. *ACM CSUR*, 54(6): 115:1–115:35.

Miller, T. 2019. Explanation in Artificial Intelligence: Insights from the social sciences. *Artif. Intell.*, 267: 1–38.

Misino, E.; et al. 2023. VAEL: Bridging Variational Autoencoders and Probabilistic Logic Programming. In *NeSy*, volume 3432 of *CEUR*, 434–435. CEUR-WS.org.

Nguyen, X. V.; et al. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *JMLR*, 11: 2837–2854.

Oussidi, A.; et al. 2018. Deep generative models: Survey. In *ISCV*, 1–8. IEEE.

Papagelis, A.; et al. 2000. GA Tree: genetically evolved decision trees. In *ICTAI*, 203. IEEE Computer Society.

Pasquale, F. 2015. *The black box society: The secret algorithms that control money and information*. HUP.

Podforelec, V.; et al. 1998. Evolutionary construction of medical decision trees. In *EMBS*, volume 3, 2–5. IEEE.

Quinlan, J. R. 1986. Induction of Decision Trees. *MACH*, 1(1): 81–106.

Quinlan, J. R. 1993. *Programs for Machine Learning C4. 5*. Elsevier.

Rajkomar, A.; et al. 2018. Scalable and accurate deep learning with electronic health records. *NPJ DM*, 1(1): 1–10.

Razavi, A.; et al. 2019. Generating Diverse High-Fidelity Images with VQ-VAE-2. In *NeurIPS*, 14837–14847.

Rivera-López, R.; et al. 2018. Construction of Near-Optimal Axis-Parallel Decision Trees Using a Differential-Evolution-Based Approach. *IEEE Access*, 6: 5548–5563.

Rivera-López, R.; et al. 2022. Induction of decision trees as classification models through metaheuristics. *Swarm Evol. Comput.*, 69: 101006.

Rokach, L.; and Maimon, O. 2005. Top-down induction of decision trees classifiers - a survey. *IEEE Trans. Syst. Man Cybern. Part C*, 35(4): 476–487.

Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, 1(5): 206–215.

Seni, G.; et al. 2010. *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Synthesis DMK. M&C.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In *NIPS*, 3483–3491.

Son, N. H. 1998. From optimal hyperplanes to optimal decision trees. *Fundamenta Informaticae*, 34(1, 2): 145–174.

Strobl, C.; et al. 2009. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological Methods*, 14(4): 323.

Tan, P.; Steinbach, M. S.; and Kumar, V. 2005. *Introduction to Data Mining*. Addison-Wesley.

Turney, P. D. 1995. Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm. *J. Artif. Intell. Res.*, 2: 369–409.

Vandewiele, G.; et al. 2016. GENESIM: genetic extraction of a single, interpretable model. *CoRR*, abs/1611.05722.

Verwer, S.; et al. 2019. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. In *AAAI*, 1625–1632. AAAI Press.

Wang, W.; et al. 2019. Topic-Guided Variational Autoencoders for Text Generation. *CoRR*, abs/1903.07137.

Wang, X.; et al. 2001. A comparative study on heuristic algorithms for generating fuzzy decision trees. *IEEE Trans. Syst. Man Cybern. Part B*, 31(2): 215–226.

Xu, L.; et al. 2019. Modeling Tabular data using Conditional GAN. In *NeurIPS*, 7333–7343.

Yang, X. 2017. Understanding the variational lower bound. *variational lower bound, ELBO, hard attention*, 13: 1–4.

Zhao, H. 2007. A multi-objective genetic programming approach to developing Pareto optimal decision trees. *Decis. Support Syst.*, 43(3): 809–826.

Zhao, H.; et al. 2020. Variational Autoencoders for Sparse and Overdispersed Discrete Data. In *AISTATS*, volume 108 of *PMLR*, 1684–1694. PMLR.