Learning Optimal Decision Trees with MaxSAT and its Integration in AdaBoost

Hao Hu\(^1\), Mohamed Siala\(^1\), Emmanuel Hebrard\(^1\), Marie-José Huguet\(^1\)
LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France\(^1\)
{hhu, siala, hebrard, huguet}@laas.fr

Abstract

Recently, several exact methods to compute decision trees have been introduced. On the one hand, these approaches can find optimal trees for various objective functions including total size, depth or accuracy on the training set and therefore. On the other hand, these methods are not yet widely used in practice and classic heuristics are often still the methods of choice. In this paper we show how the SAT model proposed by [Narodytska et al., 2018] can be lifted to a MaxSAT approach, making it much more practically relevant. In particular, it scales to much larger data sets; the objective function can easily be adapted to take into account combinations of size, depth and accuracy on the training set; and the fine-grained control of the objective function it offers makes it particularly well suited for boosting. Our experiments show promising results. In particular, we show that the prediction quality of our approach often exceeds state-of-the-art heuristic methods. We also show that the MaxSAT formulation is well adapted for boosting using the well-known AdaBoost Algorithm.

1 Introduction

Decision tree is still a very popular machine learning model for a number of reasons. A first important element in favour of decision trees is their interpretability, which, in some settings might be essential even at the cost of a lower accuracy. The wide range of efficient methods to compute decision trees is also a decisive factor to their success. Moreover, models using decision trees within ensemble methods, such as (deep) random forests, can have state of the art accuracy on a number of benchmarks [Zhou and Feng, 2017].

The most widely used techniques for computing decision trees (e.g. CART [Breiman et al., 1984], ID3 [Quinlan, 1986] and C4.5 [Quinlan, 1993]) are greedy heuristics, typically build trees from top to bottom selecting the feature yielding the highest information gain. However, it was observed early that significantly simpler trees than those computed by greedy methods might exist [Bennett, 1994]. Moreover, simpler (e.g. smaller) trees, are often more accurate on unknown data.

Recently, several exact methods have been introduced to find optimal decision trees for some combinations of criteria involving, their size (i.e., nodes), depth, and their empirical accuracy. However, despite the advantages of optimal trees, these exact algorithms have not yet replaced greedy heuristics as the most popular method. Firstly, even though great advances have been made on exact algorithms, scalability is still an issue. Moreover, one must use proxy criteria which may be more or less relevant in practice. For instance, some approaches (e.g., [Bessiere et al., 2009; Narodytska et al., 2018; Avellaneda, 2020]) minimize some size criterion, either number of nodes or maximum depth, subject to the constraint that the tree is perfectly accurate on the training set. However, this approach is often criticized as it requires consistent data sets, may entail overfitting and produces unnecessarily large trees. Indeed, greedy methods often use pruning techniques which compromise the accuracy on the training set for (often) better generalization performance. Therefore, other methods [Nijssen and Fromont, 2007; Bertsimas and Dunn, 2017; Verwer and Zhang, 2019; Verhaeghe et al., 2019; Aglin et al., 2020; Hu et al., 2019] opt instead for optimizing accuracy (on the training set) subject to constraints on the maximum depth. This approach seems better suited to most usages of decision trees since it deals naturally with noisy data sets, and the trees produced are smaller and usually more accurate. However, restricting the constraints on the tree structure to a maximum depth is very restrictive and usually entails that if the “best” tree is narrow and deep, finding it will be very costly. For instance, the SAT-encoding in [Avellaneda, 2020] can be easily adapted to use number of meaningful nodes as objective, however, the encoding will grow exponentially in the depth and therefore intractable.

In this paper, we show that a simple adaptation of the SAT model introduced by Narodytska et al. to a MaxSAT formula, offsets most of its drawbacks and makes it a very attractive method in many cases. Firstly, it makes it easy to use more practical objective functions based on accuracy. In fact, this formulation allows to use any linear combination of size, depth and accuracy as objective. Moreover, and somewhat counter-intuitively, it goes some way toward solving the scalability issue. Solving the MaxSAT formula to optimality is of course harder than solving the corresponding SAT formula.
However, dropping the constraint of perfect accuracy means that the tree, and hence the formula, can be smaller. In addition, when using state-of-the-art MaxSAT solvers [Berg et al., 2019], we observe that we can handle much larger formulas. This model finds smaller trees than the standard greedy methods in reasonable time on data sets that are out of reach for the previous SAT method from [Narodytska et al., 2018]. Finally, the MaxSAT approach can be integrated within a classical boosting method easily to improve the prediction performance. This technique still improves the scalability because individual trees of the ensemble can be smaller. Our experimental evaluation outlines the overfitting phenomenon due to the requirement of perfect accuracy in earlier approaches, and shows that the proposed approach is competitive with the state-of-the-art heuristics and exact methods.

The rest of the paper is organised as follows. In Section 2, we give the technical background of the paper. Then, we present our MaxSAT propositions in Section 3. Finally, in Section 4, we show details of our experimental results.

2 Technical Background

We present in this section a brief formal background to introduce the different notations used throughout the paper.

2.1 Classification via Decision Trees

Using for instance the fairly standard one-hot-encoding and other standard techniques, non-binary categorical features and numerical features of a dataset can be transformed into binary values. In this paper, we then focus on binary classification for data with binary features and we use similar notations to those of [Narodytska et al., 2018].

Let \( F = \{ f_1, \ldots, f_K \} \) be a set of \( K \) binary features, all of them take value in \( \{0,1\} \). The training data, denoted by \( \mathcal{E} = \{ e_1, \ldots, e_M \} \), is a set of \( M \) examples. The examples in \( \mathcal{E} \) are partitioned into \( \mathcal{E}^+ \) and \( \mathcal{E}^- \), denoting respectively positives examples and negative examples. Precisely, an example \( e_q \in \mathcal{E} \) is represented as a 2-tuple \( (x_q, c_q) \), where \( x_q \in \{0,1\}^K \) denotes the value vector for all binary features associated with the example and \( c_q \in \{0,1\} \) is the class of the example. We have \( c_q = 1 \) if \( e_q \in \mathcal{E}^+ \) and \( c_q = 0 \) if \( e_q \in \mathcal{E}^- \).

The classification problem is to learn some function \( h \) (the classifier) mapping value vectors to class values, matching as accurately as possible the actual function \( \phi \) on the training data (i.e., \( \phi(x_q) = c_q \)) and generalizes well to unseen test data. We shall use the term learner in Section 2.2 to denote the method to learn the classifier. In this paper, the function \( h \) we learn is represented by a binary and full decision tree (DT). That is, a binary tree where every internal node has exactly two children. Every internal node is associated to a feature or its negation and every leaf node is associated to a class.

2.2 Ensemble Methods

Ensemble methods is a set of methods that train multiple learners and then combine them for getting better predictions than a single classifier [Zhou, 2012]. We distinguish two families of ensemble methods: (1) Boosting methods where the learners draw from the same data set and are dependent on each other: (2) Bagging methods where the learners draw from different independent samplings and build classifiers independently from each other.

As a typical Boosting method, AdaBoost [Freund and Schapire, 1997] constructs \( T \) learners in a sequence of \( T \) iterations. Every iteration \( t \) learns a classifier \( h_t \). In each iteration \( t \), the data distribution \( D_t \) indicating the importance of each example is updated using a coefficient \( \alpha_t = \frac{1}{2} \ln \left( \frac{1 - e_t}{e_t} \right) \) where \( e_t \) is the error rate of the \( t \)th iteration.

Let \( \phi \) be the actual function, and \( Z_t \) a normalization factor. AdaBoost sets the initial distribution \( D_1(x_q) = 1/|E| \) for each example \( e_q = (x_q, c_q) \in E \). Then it updates the data distribution for the \((t+1)\)-th iteration \( D_{t+1} \) based on the \( t \)-th data distribution \( D_t \) using the equation (1):

\[
D_{t+1}(x_q) = \frac{D_t(x_q)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_q) = \phi(x_q) \\ \exp(\alpha_t) & \text{if } h_t(x_q) \neq \phi(x_q) \end{cases}
\]

(1)

Intuitively, examples that were misclassified in previous iterations will receive more attention in future iterations than those correctly classified. The final predictions are based on weighted voting where every learner \( h_t \) is associated to a weight \( \alpha_t \). That is, if \( H \) denotes the aggregated prediction, then \( H \) is calculated using equation (2).

\[
H(x_q) = 1 \text{ if } \sum_{t=1}^T \alpha_t g(h_t(x_q)) > 0 \text{ and } 0 \text{ otherwise}
\]

(2)

where \( g(0) = -1, g(1) = 1 \).

In reverse, Bagging [Breiman, 1996] constructs independent learners using different subsets (with the same size) from the original dataset. The data sampling is performed with replacement and the final prediction uses majority voting.

2.3 SAT & MaxSAT

We give a brief background on (Maximum) Boolean Satisfiability and we refer the reader to [Biere et al., 2009] for a comprehensive reference. Boolean Satisfiability (SAT) is a declarative approach to solve decision problems expressed with Boolean variables and clauses. The variables capture the unknowns of the problem and the clauses represent the constraints to satisfy. A Boolean variable \( var \) takes a value in a domain \( \text{Dom}(var) = \{0,1\} \) (or equivalently in \( \{\text{false, true}\} \)). A literal is a Boolean variable or its negation. A clause is a disjunction of literals. A value \( \text{val} \in \{0,1\} \) is assigned to a variable \( var \) if \( \text{Dom}(var) = \{\text{val}\} \). An assignment satisfies a clause if one of its literals is true. Given a set of variables \( X \) and a set of clauses defined over \( X \), the SAT problem can be defined as finding an assignment of the variables such that all the clauses are satisfied.

Maximum Satisfiability (MaxSAT) is an optimization version of SAT. The MaxSAT problem is to find an assignment of the variables maximizing the number of satisfied clauses. Here we consider the weighted partial MaxSAT problem, where clauses are separated into hard clauses (clauses that must be satisfied) and soft clauses (clauses that can be violated). Every soft clause is associated to a weight (a positive integer). In weighted Partial MaxSAT, the problem is to find an assignment that satisfies all hard clauses, and maximizes the sum of weights of soft clauses that are satisfied.
2.4 SAT-based Learning of Decision Trees

Consider the following decision problem:

\[ P(\mathcal{E}, N) : \text{Given a set of examples } \mathcal{E}, \text{ is there a full binary decision tree of size } N \text{ that classifies correctly every example in } \mathcal{E}? \]

In [Narodytska et al., 2018], a SAT model for the problem \( P(\mathcal{E}, N) \) is proposed. This model can be used as an oracle to compute a decision tree of minimal size that classifies every example. The encoding consists of three parts:

- **Part 1:** Constraints on the structure of the tree
- **Part 2:** Constraints for mapping features (respectively, classes) to internal nodes (respectively, leaf nodes)
- **Part 3:** Constraints for correctly classifying every example in the training set

We refer the reader to the original paper for the full model. In the following, we describe a part of the model needed to control the depth of the tree in this section. Note that classifying correctly every example \( e_q \in \mathcal{E} \) is true only if \( e_q \) is correctly classified by the tree. Linking the \( b_q \) variables with part 3 of the SAT model is done with the following hard clauses:

\[
\forall e_q \in \mathcal{E}^+, \forall i \in [1, N], b_q \rightarrow (v_i \land \neg c_i \lor \bigvee_{r=1}^K d^{\sigma(r,q)}_{r,i}) \quad (5)
\]

\[
\forall e_q \in \mathcal{E}^-, \forall i \in [1, N], b_q \rightarrow (v_i \land c_i \lor \bigvee_{r=1}^K d^{\sigma(r,q)}_{r,i}) \quad (6)
\]

Finally, in order to model the objective of maximizing correctly classified examples, each literal \( b_q \) is declared as a soft clause. Clearly, the number of satisfied soft clauses is equal to the number of correctly classified examples.

3.1 MaxSAT Model to Learn Decision Trees

We propose in this section a MaxSAT encoding for the optimisation problem \( P^*(\mathcal{E}, N) \). That is, given a set of examples \( \mathcal{E} \), find a full binary decision tree of size \( N \) that maximises the number of examples in \( \mathcal{E} \) that are correctly classified.

All the constraints from part 1 and part 2 of the SAT model discussed in Section 2.4 are kept as hard clauses. Then, for classifying the examples, we introduce one Boolean variable \( b_q \) for every example \( e_q \in \mathcal{E} \). The encoding consists of three parts:

- **Part 1:** Constraints on the structure of the tree
- **Part 2:** Constraints for mapping features (respectively, classes) to internal nodes (respectively, leaf nodes)
- **Part 3:** Constraints for correctly classifying every example in the training set

Let \( e_q \in \mathcal{E}^+, \) and let \( \sigma(r,q) \in \{0,1\} \) be the value of the feature \( f_r \) for \( e_q \). The following constraints are used to forbid \( e_q \) to be classified by a leaf node \( i \) (\( i \in [1, \ldots, N] \)) associated with a negative class:

\[
v_i \land \neg c_i \lor \bigvee_{r=1}^K d^{\sigma(r,q)}_{r,i} \quad (3)
\]

The same idea is used for every example \( e_q \in \mathcal{E}^- \):

\[
v_i \land c_i \lor \bigvee_{r=1}^K d^{\sigma(r,q)}_{r,i} \quad (4)
\]

3.2 Constraints for Controlling Depth

A same tree size might clearly lead to different tree topologies. From Figure 1, we can see two extreme situations using the same size 7: a fully unbalanced binary tree (left), and a complete (balanced) binary tree (right). Having a way to control the topology of the tree, via both the depth and the size, is a desired property to have, when learning decision trees. We show how to control the depth of the tree in this section. Note that we count the depth of binary tree from root as depth 0.

![Figure 1: Two Binary Trees of the Same Size with Different Depths](image)

We introduce a Boolean variable \( depth_{i,t} \) that is true iff the node \( i \) is in depth \( t \). Observe that \( t \) can only be in \( \lfloor \log(i + 1) \rfloor - 1, \lfloor (i - 1)/2 \rfloor \) for every node \( i \) of the tree. The maximum allowed depth of the tree is encoded in the following constraint:

\[
\sum_{t \in \lfloor \log(i + 1) \rfloor - 1, \lfloor (i - 1)/2 \rfloor} depth_{i,t} = 1, \quad \text{for } i = 1, \ldots, N \quad (7)
\]

If node \( i \) is in depth \( t \), and node \( j \) is a child of node \( i \), then node \( j \) must be in depth \( t + 1 \).

\[
depth_{i,t} \land l_{ij} \rightarrow depth_{j,t+1}, \quad \text{for } i = 1, \ldots, N, j \in LR(i) \quad (8)
\]

\[
depth_{i,t} \land r_{ij} \rightarrow depth_{j,t+1}, \quad \text{for } i = 1, \ldots, N, j \in RR(i) \quad (8)
\]

By classifying every example in the training set, the previous SAT model naturally introduces overfitting. This issue can be avoided by considering the maximisation problem:

\[ P^*(\mathcal{E}, N) : \text{Given a set of examples } \mathcal{E}, \text{ find a full binary decision tree of size } N \text{ that maximises the number of examples in } \mathcal{E} \text{ that are correctly classified.} \]

In the following we show how the problem \( P^*(\mathcal{E}, N) \) can be effectively modelled using MaxSAT. Next, we show adaptations of the model to take care of the maximum/exact depth of the tree and to model a relaxation of the problem where \( N \) is an upper bound on the number of nodes.
If the maximum depth given is $U$, then all possible nodes in depth $U$ must be leaf nodes.

$$\text{depth}_{i,U} \to v_i, \text{ for } i \in [2U, \min(2^U+1-1, N)]$$  \quad (9)

The following constraint can be added if $U$ is given as an exact depth instead of an upper bound.

$$\min(2^U+1-1, N) \bigvee_{i=2U} \text{depth}_{i,U} = 1$$  \quad (10)

### 3.3 Constraints for Controlling Tree Size

We show that the previous encoding can be easily adapted to build a tree with an upper bound on the size of the tree instead of the exact size. Consider firstly the relationship between the size and the depth in a full binary tree. For a maximum depth $U$, the upper bound of the size is $2^{U+1} - 1$. If $U$ is set as exact depth, we additionally get the lower bound for the size as $2U + 1$. Observe also that the size of a full and binary decision tree can only be an odd number starting from 3.

Suppose that $N$ is an upper bound on the number of nodes. We introduce a set of Boolean variables $m_i$ (where $i \in \{3, 5, 7, \ldots, N\}$) to indicate if at least $i$ nodes are used to construct the tree. Recall that we need at least 3 nodes. We therefore need to enforce $m_3$ to be true. If at least $i+2$ nodes are used then at least $i$ nodes are used:

$$m_{i+2} \to m_i, \text{ for all } i \in [1, N-2]$$  \quad (11)

We then apply the following simple rule to adapt each constraint (parts 1 & 2, the new constraints of part 3, and the depth constraints): We look at each clause $C$ from the encoding separately, and consider $j$, the largest node index used in $C$. We simply replace the clause $C$ by:

$$m_j \to C \text{ if } j \text{ is odd}$$
$$m_{j+1} \to C \text{ if } j \text{ is even}$$  \quad (12)

That is, if $j$ is odd (respectively even) and at least $j$ (respectively $j+1$) nodes are used, then the clause $C$ must hold.

### 3.4 Integration in AdaBoost

In this section, we explain how the framework of MaxSAT is well adapted to implement AdaBoost. The MaxSAT model used to learn the decision tree at iteration $t$ is identical to the previous model (in which one can control the size and the depth) except for the weight associated to each soft clause $b_q$. Indeed, we use weighted partial MaxSAT to approximate the data distribution $D_t$ by associating every soft clause $b_q$ to a positive integer weight $w_q^t$. The weighted voting follows the original AdaBoost algorithm in Equation (2). We set all weights at the first iteration with the value 1 as initial distribution indicating the equal importance of the examples. Then, the value of $w_q^{t+1}$ is calculated based on $w_q^t$ in two steps as follows. Firstly, we update and normalize the weights:

$$w_q^{t+1} = \frac{w_q^t \ast \text{factor}_q}{\sum_{q=1}^{M} (w_q^t \ast \text{factor}_q)}$$  \quad (13)

where $\text{factor}_q^t$ is the updating factor based on prediction:

$$\text{factor}_q^t = \begin{cases} \exp(-\alpha_t) & \text{if } h_t(x_q) = \phi(x_q) \\ \exp(\alpha_t) & \text{if } h_t(x_q) \neq \phi(x_q) \end{cases}$$  \quad (14)

Secondly, we discretize the weight $\hat{w}_q^{t+1}$ as follows:

$$\hat{w}_q^{t+1} = \text{round}(\frac{w_q^{t+1}}{\min_{i\in\{1,\ldots,M\}}(\hat{w}_i^{t+1})})$$  \quad (15)

### 4 Experimental Results

We perform three experiments on datasets from CP4IM\(^1\). The first experiment aims at highlighting the overfitting behaviour of the SAT approach. In the second experiment we compare the performance with the state of the art heuristic (CART) and exact (DL8.5) methods. Finally, we evaluate the MaxSAT implementation of Boosting and Bagging. The datasets are binaryized with the one-hot-encoding. We ran all experiments using Xeon E5-2695 v3 @ 2.30GHz CPU and running xUbuntu 16.04.6 LTS. Our code source and datasets are available online at https://gepgitlab.laas.fr/hhu/maxsat-decision-trees.

#### 4.1 The Overfitting Phenomenon

The first set of experiments aims to show the existence of the overfitting behaviour for the SAT method of learning optimal size decision trees that classify every example in the training set. For this, we learn decision trees with our MaxSAT encoding by increasing the size of the trees starting from 3 until we find the size that classifies correctly every example in the training set. The final decision tree obtained with the proposed MaxSAT model corresponds to the SAT model of [Narodytska et al., 2018]. We use the hold-out method to split training set and testing set for all datasets used. Following [Narodytska et al., 2018], we choose (only in this experiment) three (small) different splitting ratios $r = \{0.05, 0.1, 0.2\}$ to generate training sets. And remaining examples are used for testing. This process is repeated 10 times using different randomisation seeds. We use the RC2 MaxSAT solver [Ignatiev et al., 2019]. In the context of these experiments, the solver is left with no limit until it finds the optimal solution (in terms of training accuracy).

In Figure 2, we report the average training accuracy and testing accuracy for one dataset using the different sampling ratios. We can see that the training accuracy increases with the tree size, until reaching a perfect classification. Notice that the tree that classifies every example has the same size as the one using the original SAT approach of [Narodytska et al., 2018]. The testing accuracy shown in Figure 2(b) shows that the perfect tree overfits the training data, since smaller trees, while less accurate on the training set have a better testing accuracy. This phenomenon is not as marked for every dataset, however, we almost systematically observe a plateau whereby the testing accuracy stays constant at best while the training accuracy increases.

\(^1\)https://dtai.cs.kuleuven.be/CP4IM/datasets/
In this section, we compare the prediction accuracy of the decision trees learnt by MaxSAT with CART.

4.2 Comparison with Different Methods

We do not report exact run time of CART cause it takes only few seconds. In terms of training and testing accuracy, Table 1 shows that the trees learnt via MaxSAT encoding are competitive with both heuristic and exact methods in prediction performance. Although MaxSAT model can not always report optimality, it is close to the optimal solution obtained by DL8.5. In addition, DL8.5 needs massive memory for deep trees, while the MaxSAT model does not. For instance, for a maximum depth of 5, DL8.5 runs out of memory on 6 datasets, even when lifting the limit to 50GB.

4.3 Boosting via MaxSAT

In order to show the impact of our AdaBoost implementation with MaxSAT, we compare it to bagging, a single learner using MaxSAT, and AdaBoost based on CART. We fix the parameters (size, depth, timeout) for every learner to be the same. In particular, we use the MaxSAT model with an upper bound for the depth (i.e., DT(max)) using the Loandra solver and a timeout of 15 minutes. The number of learners (for boosting and bagging) is set to be 21 as reasonable. We chose datasets where the MaxSAT models do not perform well from the results in Table 1. We use the hold-out method with ratio \( r = 0.8 \) to split each dataset into training and testing set, which is repeated with 10 different randomisation seeds.

The results are shown in Table 2 (where BS stands for the AdaBoost model via MaxSAT, BSH stands for the AdaBoost model based on CART, and BG stands for the Bagging model). The accuracy values are shown in percent. The best values between the three methods are in bold. This table clearly shows that the Boosting via MaxSAT does improve the training accuracy in almost all the cases with respect both to bagging and to a single learner. The boosted trees via MaxSAT generalises very well and are competitive with those based on CART.

5 Conclusion

We propose an exact method for learning decision trees based on the MaxSAT framework. Our initial motivation behind this work was to avoid the overfitting issue with the previous SAT encoding and to address the scalability issue. The declarative nature of MaxSAT made it possible to additionally control the size and the depth of the learnt trees. The empirical results show that our approach learns good quality decision trees compared to heuristics method and the recent state-of-the-art. We also show that our MaxSAT-based method is well suited for applying boosting techniques.

In the future, it would be interesting to guide the branching heuristic using the information gain coefficient. Furthermore, we will explore the integration of our method into the Random Forest framework [Breiman, 2001].

Acknowledgements

The authors would like to thank Nina Narodytska for kindly sharing the source code of the original SAT model.
References


