MID: A New Strategy for Learning Optimal Decision Trees on Continuous Data

Antonio Dal Maso[1,*], Harold Kiossou[2] and Siegfried Nijssen[2]

[1]Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Turin, Italy
[2]ICTEAM, UCLouvain, Place Sainte-Barbe 2, bte L5.02.01, B-1348 Louvain-la-Neuve, Belgium

Abstract
Existing Optimal Decision Tree (ODT) algorithms, which are designed to find the best decision tree on training data, predominantly work on binary features and require the use of discretization algorithms to preprocess continuous ones. In most cases, these techniques make it unfeasible in practice to find an ODT. We propose a new approach for learning decision trees on continuous data by combining a new discretization algorithm, MID, with ODT algorithms. Given infinite time, this allows the ODT algorithm to find an ODT, but if interrupted early, the approach still produces a good decision tree. Experiments on 14 datasets provide evidence of the effectiveness of this method.

Keywords
Supervised Discretization, Impurity Metrics, Optimal Decision Trees

1. Extended Abstract

Decision tree learning algorithms are among the most widely used machine learning methods today, thanks to the predictive power and interpretability of the learned models. Since finding an optimal decision tree (that minimizes the training error) under constraints is NP-hard, greedy algorithms like CART [1] and C4.5 [2] have long been the preferred methods for this task. These algorithms are fast and can handle all types of data, from binary to continuous, without the need for preprocessing. However, they often find suboptimal trees that do not minimize error across the entire dataset. In recent years, advancements in optimization solvers and novel algorithmic concepts have led to the development of new methods for inferring decision trees that are optimal on training data (ODTs). Various approaches have been proposed, including mixed integer programming [3, 4, 5, 6], constraint programming [7], and SAT solvers [8]. Among these, dynamic programming methods like DL8, DL8.5, and MurTree [9, 10, 11] have emerged as particularly effective, offering both accuracy and efficiency in depth-constrained settings. These methods leverage advanced optimization techniques to explore the search space more thoroughly than traditional greedy algorithms. As these techniques are typically designed for binary datasets, they often require continuous features to be discretized before the learning process – a step that can significantly impact the quality and efficiency of the resulting decision tree. As a consequence, there is no assurance that such a tree is optimal with respect to the training data.

A number of discretization techniques have been proposed in the literature. Straightforward techniques, such as equal-width and equal-frequency binning, often lead to suboptimal trees because they fail to capture the underlying structure of the data effectively. More sophisticated methods, like the MDLP discretizer [12] or ChiMerge [13], use heuristics to select the most informative cut-points, offering better results in practice.

All these discretization techniques share the following characteristics: (1) they operate on each continuous feature individually; (2) they offer parameters to determine the number of discretized features generated per continuous feature. In general, depending on the choice of parameters, these

Discovery Science - Late Breaking Contributions 2024
*Corresponding author.
s311444@studenti.polito.it (A. Dal Maso); harold.kiossou@uclouvain.be (H. Kiossou); siegfried.nijssen@uclouvain.be (S. Nijssen)
0000-0001-6972-9885 (H. Kiossou); 0000-0003-2678-1266 (S. Nijssen)

discretizers can either produce too many binary features, increasing the number of trees that optimal algorithms have to consider, or oversimplify the data, leading to a loss of accuracy and learning trees that are far from optimal.

In this paper, we introduce the Minimum Impurity Discretizer (MID) as a new technique to address these weaknesses. MID distinguishes itself by combining several characteristics: (1) it generates a set of binary features using an iterative approach that starts with an empty set and adds new features to it at each iteration; (2) it considers all input continuous features jointly when creating the new binary ones, allowing some features to be discretized more finely than others; (3) it can be used with ODT algorithms in an anytime manner, where the learner is run iteratively over data of increasing dimensionality, and the search process stops when resources are no longer available; (4) it allows one to find good decision trees even when a small set of binary features is used for training.

Algorithm 1 MID training process

```
Require: Dataset $X$, target variable $y$, number of output binary features $N$
 1: $sorted\_candidates\_per\_feature \leftarrow$ empty list [ ]
 2: for $x$ in columns of $X$ do
 3:     $x$ and $y$ are sorted based on the values of $x$
 4:     $candidates \leftarrow$ compute_and_rank_feature_candidates($x, y$)
 5:     $candidates$ is appended to $sorted\_candidates\_per\_feature$
 6: $thresholds \leftarrow$ get_best_thresholds($sorted\_candidates\_per\_feature$,  $N$)
 7:
 8: Procedure get_best_thresholds($candidates\_per\_feature$, $N$)
 9:     $thresholds \leftarrow$ empty list [ ]
10:     while length($thresholds$) < $N$ do
11:         $idx, max\_impurity\_gain \leftarrow -1, -\infty$
12:         $empty \leftarrow 0$
13:         for ($i, list$) in enumerate($candidates\_per\_feature$) do
14:             if length($list$) > 0 then
15:                 if $value < list[0][1]$ then
16:                     $idx \leftarrow i$
17:                     $max\_impurity\_gain \leftarrow list[0][1]$
18:                     $best\_candidate \leftarrow list[0][0]$
19:             else
20:                 $empty \leftarrow empty + 1$
21:         if $empty ==$ length($candidates\_per\_feature$) then
22:             break
23:         the tuple ($idx, best\_candidate$) is appended to $thresholds$
24:         the head of $candidates\_per\_feature[idx]$ is popped
25:     return $thresholds$
```

Algorithm 1 outlines how MID computes thresholds for discretizing a continuous dataset. First, each feature is considered individually: for each of them, potential thresholds are ranked using an impurity metric, such as entropy or the Gini index (line 4). Each threshold in this list is the one that, if applied to further split the feature's range, would minimize the impurity of the class labels, under the assumption that all higher-ranked thresholds have been applied. The list *candidates*, introduced at line 4, consists of tuples in the form (*candidate_threshold*, *impurity_gain*), where *impurity_gain* quantifies the reduction in impurity achieved by applying *candidate_threshold*.

The lists associated with the individual features are then merged together, ensuring that the thresholds retain their relative order from the original rankings. This is done by `get_best_thresholds` (lines 8-25), which also extracts the top N thresholds from the resulting global ranking. The procedure iteratively compares the top thresholds from the lists of all features, selecting and removing the best threshold

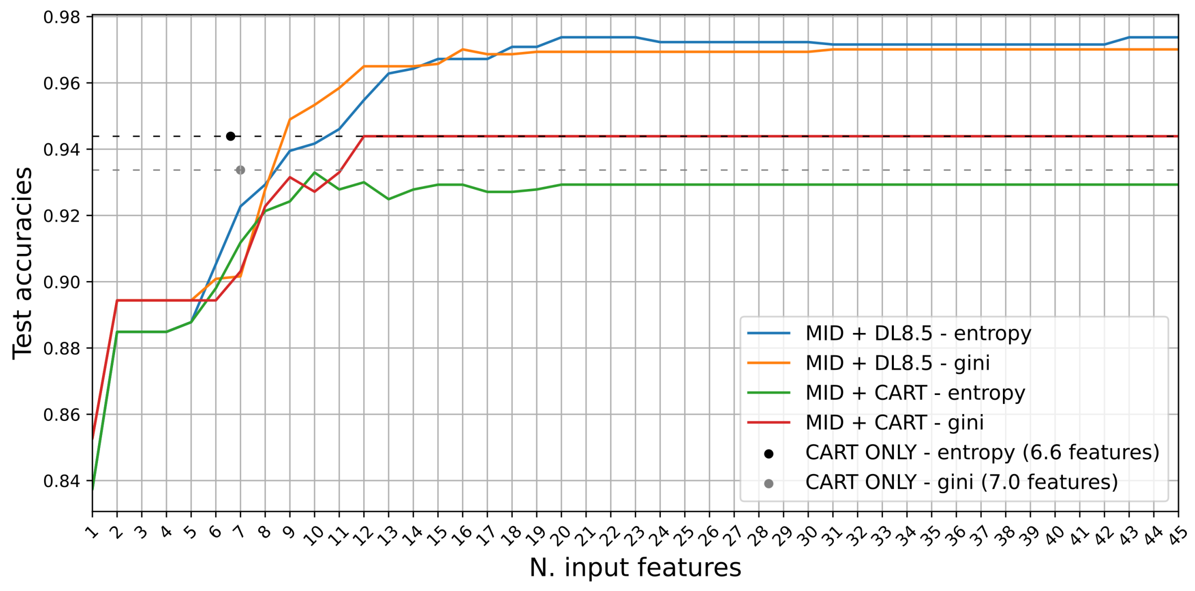

Figure 1: Average test accuracies achieved by 10-fold cross validation for DL8.5 and CART on the banknote dataset using a maximum depth of 3. The numbers of features reported for CART on the continuous data are not integers because they are average values as well.

at each step. This process continues until N binary features are retrieved (line 10) or until no more thresholds are available (lines 21-22). Each threshold in the global ranking is the one that would reduce the impurity of the class labels the most if applied to further discretize the corresponding continuous feature (lines 15-18). We consider this candidate as the optimal choice to continue the discretization.

At this point, a set of N binary features representing the dataset can be obtained by selecting the top N thresholds from the global ranking. Let X_1 and X_2 be two sets of thresholds containing N_1 and N_2 elements, respectively. By construction, if $N_1 < N_2$, then $X_1 \subseteq X_2$, meaning that the latter set of binary features is a superset of the former one. If an optimal decision tree learner is trained on both datasets, its performance on X_2 will be better than or equal to that on X_1. MID makes it straightforward to represent the observations in a dataset using a small number of binary features. Moreover, additional features can easily be added if the user has more availability in terms of time and resources. Intuitively, whenever a new binary feature is needed, MID selects both the best feature to split and the optimal cut point according to the impurity minimization heuristic.

As anticipated before, MID allows an ODT algorithm to be run repeatedly on input data with growing dimensionality. If the process is run without a time limit, an optimal decision tree can be found, but the search process can also be interrupted at any time for a smaller number of features. It is important to notice that the way in which MID operates is particularly suitable for this approach, as producing more than one set of binary features starting from the same continuous dataset only requires to train MID a single time. Moreover, the difference between the binary datasets used in two consecutive iterations consists in only one feature. Thus, repeating the discretization process multiple times only adds a small overhead in terms of runtime. An extensive analysis of the pseudocode of MID and its characteristics is available at https://github.com/antoniodalmaso/MID, together with the Python implementation used in the experiments and all associated results.

We now discuss the experiments. MID was evaluated using 14 datasets from the UCI Machine Learning Repository [14], comparing its performance with that of other discretization methods and greedy algorithms. Our experiments revealed that MID, when combined with DL8.5, consistently achieves higher accuracy than both CART (trained on both binary and continuous data) and DL8.5 paired with other discretizers, such as MDLP[1] or equal-frequency binning. For example, Figure 1 shows

[1]We used the implementation available at https://github.com/navicto/Discretization-MDLPC.

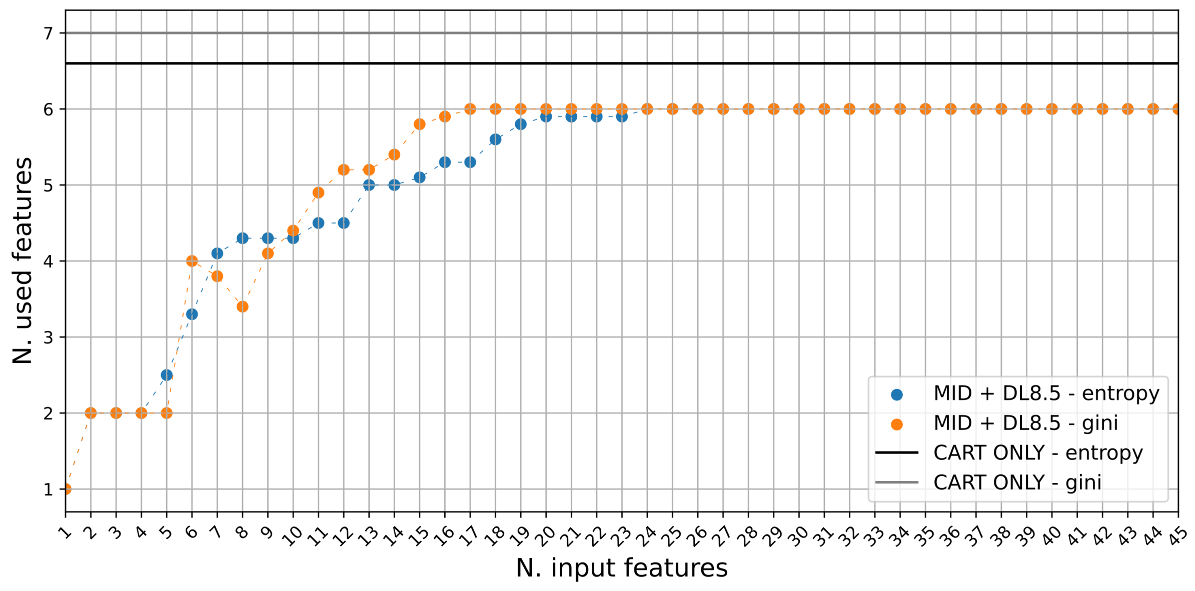

Figure 2: Number of unique binary features used in the decision trees found by DL8.5 as a function of the number of features fed to it. The numbers of used features are not all integers because this results are obtained using 10-fold cross validation. Banknote dataset, maximum depth of 3.

how the test accuracies obtained by DL8.5 and CART on the banknote dataset with a maximum depth of 3 vary when different numbers of binary features are used for the training. The x-axis represents the number of binary features fed to the learners. Please note that this axis hence does not indicate how many features the resulting decision tree model uses. The accuracies achieved by CART on continuous data are added as dots using the number of unique thresholds tested in the resulting trees as x-coordinates. The results show that, initially, the test accuracies of CART and DL8.5 sharply increase with the number of binary features produced by MID. Both then reach a plateau as the number of features increases. This behavior can be observed in most of the experiments across all datasets, and it suggests that one could determine how many binary features to use by iteratively adding them until there is no significant change in the test accuracy (i.e., when the accuracy stops improving or decreases). Furthermore, when comparing the number of features actually used in the final trees, as reported in Figure 2 for the banknote dataset, it appears that MID and DL8.5 use less features (at most 6) than CART on the continuous data. Thus, they manage to achieve higher accuracies while producing smaller trees. Experiments on the remaining datasets give results consistent with those described above. Moreover, in terms of run time, the proposed approach is performant: DL8.5's performance is very good when its number of input features is small, as is the case here.

In conclusion, we introduced MID, a heuristic-based, supervised, multivariate discretizer designed to enhance the performance of optimal decision tree algorithms like DL8.5 on continuous datasets. As a future work, it could be interesting to develop an MDL criterion similar to the one used by MDLP to make the finetuning of MID easier for the user, and to explore the effect of other impurity metrics on its performance.

Acknowledgments

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

References

[1] L. Breiman, Classification and regression trees, Routledge, 2017.

[2] J. R. Quinlan, C4. 5: programs for machine learning, Elsevier, 2014.

[3] S. Aghaei, A. Gómez, P. Vayanos, Strong optimal classification trees, Operations Research (2024). doi:10.1287/opre.2021.0034.

[4] D. Bertsimas, J. Dunn, Optimal classification trees, Machine Learning 106 (2017) 1039–1082.

[5] S. Verwer, Y. Zhang, Learning optimal classification trees using a binary linear program formulation, in: Proceedings of the AAAI conference on artificial intelligence, volume 33, 2019, pp. 1625–1632. doi:10.1609/aaai.v33i01.33011624.

[6] J. J. Boutilier, C. Michini, Z. Zhou, Shattering inequalities for learning optimal decision trees, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2022, pp. 74–90. doi:10.1007/978-3-031-08011-1_7.

[7] H. Verhaeghe, S. Nijssen, G. Pesant, C.-G. Quimper, P. Schaus, Learning optimal decision trees using constraint programming, Constraints 25 (2020) 226–250.

[8] N. Narodytska, A. Ignatiev, F. Pereira, J. Marques-Silva, Learning optimal decision trees with sat, in: International Joint Conference on Artificial Intelligence 2018, Association for the Advancement of Artificial Intelligence (AAAI), 2018, pp. 1362–1368. doi:10.24963/ijcai.2018/189.

[9] G. Aglin, S. Nijssen, P. Schaus, Learning optimal decision trees using caching branch-and-bound search, in: Proceedings of the AAAI conference on artificial intelligence, volume 34, 2020, pp. 3146–3153.

[10] E. Demirović, A. Lukina, E. Hebrard, J. Chan, J. Bailey, C. Leckie, K. Ramamohanarao, P. J. Stuckey, Murtree: Optimal decision trees via dynamic programming and search, Journal of Machine Learning Research 23 (2022) 1–47.

[11] S. Nijssen, E. Fromont, Mining optimal decision trees from itemset lattices, in: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, 2007, pp. 530–539. doi:10.1145/1281192.1281250.

[12] U. M. Fayyad, K. B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Ijcai, volume 93, Citeseer, 1993, pp. 1022–1029.

[13] R. Kerber, Chimerge: Discretization of numeric attributes, in: Proceedings of the tenth national conference on Artificial intelligence, 1992, pp. 123–128.

[14] M. Kelly, R. Longjohn, K. Nottingham, The uci machine learning repository, last accessed: May 28, 2024. URL: https://archive.ics.uci.edu.