Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)
Demonstrations Track

PyDL8.5: a Library for Learning Optimal Decision Trees

Gaël Aglin , **Siegfried Nijssen** and **Pierre Schaus**
ICTEAM, UCLouvain
{firstname.lastname}@uclouvain.be

Abstract

Decision Trees (DTs) are widely used Machine Learning (ML) models with a broad range of applications. The interest in these models has increased even further in the context of Explainable AI (XAI), as decision trees of limited depth are very interpretable models. However, traditional algorithms for learning DTs are heuristic in nature; they may produce trees that are of suboptimal quality under depth constraints. We introduce PyDL8.5, a Python library to infer depth-constrained Optimal Decision Trees (ODTs). PyDL8.5 provides an interface for DL8.5, an efficient algorithm for inferring depth-constrained ODTs. The library provides an easy-to-use scikit-learn compatible interface. It cannot only be used for classification tasks, but also for regression, clustering, and other tasks. We introduce an interface that allows users to easily implement these other learning tasks. We provide a number of examples of how to use this library.

1 Introduction

Machine learning models are increasingly used in a wide range of applications. However, an increasing concern is the interpretability of machine learning models. Whether or not human experts can understand a model can for instance be important to avoid ethical problems [Craven and Shavlik, 2014; Ribeiro *et al.*, 2018; Ribeiro *et al.*, 2016a; Ribeiro *et al.*, 2016b; Ignatiev *et al.*, 2019]. Decision trees have gained interest in recent years for this reason, since they can be interpreted as rules that are interpretable by domain experts.

An important parameter for the interpretability of a decision tree is the depth of the tree. Deeper trees contain more tests and more rules, and hence are often harder to interpret.

The most well-known algorithms such as CART [Breiman *et al.*, 1984] and C4.5 [Quinlan, 1993] use a greedy approach to learn DTs. They select iteratively the best feature to split on based on a heuristic (information gain, gini index) and continue this splitting process until a desired quality (eg., classification accuracy) is obtained or a desired depth is reached.

However, this greedy process has disadvantages. It has been shown that trees found using greedy algorithms are suboptimal when also the depth is taken into account: greedy trees that are sufficiently accurate are sometimes unnecessarily deep, and depth-constrained trees are not sufficiently accurate [Bertsimas and Dunn, 2017]. Moreover, to learn trees for other tasks than classification, it is often necessary to develop new heuristics [Blockeel *et al.*, 1998], making it harder to solve such tasks.

For this reason, in recent years researchers [Nijssen and Fromont, 2007; Bertsimas and Dunn, 2017; Verwer and Zhang, 2019; Hu *et al.*, 2019; Verhaeghe *et al.*, 2019; Aglin *et al.*, 2020] have studied algorithms for learning *Optimal Decision Trees* (ODTs): trees that under well-defined constraints, such as on depth, achieve the highest possible score on training data.

The main challenge in finding ODTs is the NP-hardness of finding ODTs. Good search algorithms are needed to make solving this task feasible. We recently showed that an algorithm that we proposed, DL8.5, obtains the best performance on a wide range of test cases [Aglin *et al.*, 2020]. Indeed, on commonly used UCI datasets DL8.5 computes ODTs within seconds, making its use in an interactive demo possible.

In this paper, we introduce PyDL8.5, an open source Python library that implements DL8.5 in an efficient and extendible manner. Compared to other systems for finding ODTs, it offers these advantages: (1) it offers better computational performance, as shown in our recent study [Aglin *et al.*, 2020]; (2) it is easy to use, as it is fully compatible with scikit-learn; (3) it can easily be extended to solve other tree learning problems than classification problems.

To deal with other learning problems, we implemented DL8.5 such that it works for any optimization criterion that is *additive*. Using our library, a user can express her optimization criterion using any Python library of choice; the optimization criterion does not need to be linear or integer, as required in alternative methods based on MIP and SAT solvers.

This paper is organized as follows. In the next section, we summarize DL8.5. Then, we present the PyDL8.5 library and show examples of how to use it on some ML tasks.

2 Learning ODTs using DL8.5

Given a binary training set $\mathcal{D}$, and two parameters $maxdepth$ and $minsup$, the problem that is solved by the DL8.5 algorithm is to find the decision tree $\arg\min_{T \in \mathcal{T}} f(T)$, where

- $\mathcal{T}$ represents all decision trees of depth $\leq maxdepth$ in

which each leaf covers at least $minsup$ examples of the training data $\mathcal{D}$;

- $f(T)$ is a scoring function that is additive over the leafs of the tree T, that is, f can be written as $f(T) = \sum_{\ell \in leafs(T)} g(\ell)$, where $g(\ell) \geq 0$ is a function that evaluates the quality of each leaf ℓ, and $g(\ell)$ is independent of the order of the tests leading to leaf ℓ.

In [Aglin *et al.*, 2020] DL8.5 was introduced for the problem of classification; a scoring function $g(\ell)$ was used that evaluates the misclassification error of a leaf, that is, the number of instances covered by a leaf that do not belong to the majority class of that leaf. However, the algorithm is correct for other additive scoring functions as well.

DL8.5's search algorithm relies on a mix of Dynamic Programming and Branch-and-Bound search. It recursively explores all possible splits and selects the split with the lowest score. Since different sequences of splits may select the same set of instances in the data, the same subset of data may be encountered multiple times during the search. DL8.5 uses a caching system to reuse results. To prune the search space, DL8.5 uses a bounding system. When a subtree is found, its score is used as an upper-bound to restrict the quality of future subtrees. Here DL8.5 exploits the additive nature of the scoring function to prune a right-hand subtree when the left-hand subtree provides an error greater or equal to the upper-bound.

Given its branch-and-bound nature, providing an initial upper-bound on the quality of a tree can hence help the search. This upper-bound is an optional parameter.

3 PyDL8.5

Architecture. Given the popularity of Python in data science and AI, we implemented PyDL8.5 as a Python 3 library. The core of the algorithm is written in C++ and uses 64-bit vector operations to improve the performance. PyDL8.5 implements the `fit`/`predict` interface of the popular scikit-learn library [Pedregosa *et al.*, 2011] to make it easy to use in combination with scikit-learn. An important component of DL8.5 is the scoring function used to evaluate the leafs of a tree. For the most common scoring functions, a fast implementation in C++ is provided. This is important as the function is called very often. Users can also write a scoring function in Python, although such functions may slow down the execution. The library is hosted on PyPI[1] and the source code is available at https://github.com/aia-uclouvain/pydl8.5. The following examples demonstrate how easily PyDL8.5 can be used to implement many different ODT learning tasks.

Example Task 1: Shallow Classifiers. Listing 1 shows the code needed to train an ODT classifier and predict on unseen data. Note the simple integration in scikit-learn. The score minimized by default by `DL85Classifier` is the misclassification error. In the listings 2 and 3, we will omit the code of Listing 1 from line 1 to line 8.

```
1 import numpy as np
2 from sklearn.model_selection import train_test_split
3 from dl85 import DL85Classifier
4 # read the dataset and split into features and targets
5 dataset = np.genfromtxt("anneal.txt")
6 X, y = dataset[:, 1:], dataset[:, 0]
7 # split the dataset into training and test sets
8 X_train,X_test,y_train,y_test = train_test_split(X, y,
      test_size=0.2)
9 # initialize the classifier, train and predict
10 clf = DL85Classifier(max_depth=3, min_sup=5)
11 clf.fit(X_train, y_train)
12 y_pred = clf.predict(X_test)
```

Listing 1: Code snippet to train a classifier

Example Task 2: Predictive Clustering. Predictive clustering is an unsupervised task in which one aims to identify clusters of good quality in the leafs of the tree and the tree can be interpreted as a description of the clusters [Blockeel *et al.*, 1998]. Listing 2 shows how to implement this using a custom scoring function `error` that calculates the sum of euclidean distances from each point in a cluster to the centroid. Note that no heuristic is needed, that this function is nonlinear, and is written using NumPy code itself. A `leaf_value` function is provided to determine the labels put in the leafs of the tree.

```
1 from dl85 import DL8Predictor
2 from sklearn.neighbors import DistanceMetric
3 eucl_dist = DistanceMetric.get_metric('euclidean')
4 # user-defined scoring function
5 def error(tids):
6   X_subset = X_train[list(tids),:]
7   centroid = np.mean(X_subset, axis=0)
8   distances = eucl_dist.pairwise(X_subset, [centroid])
9   return float(sum(distances))
10 # user-defined labels in the leaf
11 def leaf_value(tids):
12   return np.mean(X_train.take(list(tids)))
13 # initialize the search and run it
14 clf = DL85Predictor(max_depth=3, min_sup=5,
      error_function=error, leaf_value_function=
      leaf_value)
15 clf.fit(X_train)
16 predicted = clf.predict(X_test)
```

Listing 2: Code snippet of user-defined clustering

Example Task 3: 100% Accurate Classifiers. A recent publication studied the problem of finding the shallowest tree with perfect accuracy on training data (100%); it solved this problem using a SAT solver [Avellaneda, 2020]. The following code shows how this problem can be solved using PyDL8.5. We found that this simple piece of code typically solves the same problem more rapidly. In this code, we exploit the fact that we can specify a maximum allowed error.

```
1 maxdepth = 1
2 while True:
3   clf = DL85Classifier(max_depth=maxdepth, min_sup=2,
        max_error=1)
4   clf.fit(X, y)
5   if clf.error_ == 0:
6     break
7   maxdepth += 1
8 y_pred = clf.predict(X_test)
```

Listing 3: Code snippet to train the shallowest 100% accurate DT

We believe that our library can be used in many applications. Questions such as how well ODTs work in ensembles, regression, probability estimation, multi-target prediction, and so on, remain currently open, but can now be studied easily using PyDL8.5.

Acknowledgements

This work was supported by Bpost.

[1] https://pypi.org/project/dl8.5/

References

[Aglin *et al.*, 2020] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[Avellaneda, 2020] Florent Avellaneda. Efficient inference of optimal decision trees. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[Bertsimas and Dunn, 2017] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[Blockeel *et al.*, 1998] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), 1998*, pages 55–63, 1998.

[Breiman *et al.*, 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[Craven and Shavlik, 2014] Mark W Craven and Jude W Shavlik. Learning symbolic rules using artificial neural networks. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 73–80, 2014.

[Hu *et al.*, 2019] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In *Advances in Neural Information Processing Systems*, pages 7265–7273, 2019.

[Ignatiev *et al.*, 2019] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva. Abduction-based explanations for machine learning models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1511–1519, 2019.

[Nijssen and Fromont, 2007] Siegfried Nijssen and Elisa Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539, 2007.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[Ribeiro *et al.*, 2016a] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[Ribeiro *et al.*, 2016b] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[Ribeiro *et al.*, 2018] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[Verhaeghe *et al.*, 2019] Hélene Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. In *The 25th International Conference on Principles and Practice of Constraint Programming (CP2019)*, 2019.

[Verwer and Zhang, 2019] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1625–1632, 2019.