

# ODTlearn: A Package for Learning Optimal Decision Trees for Prediction and Prescription

Patrick Vossler\*

Sina Aghaei

Nathan Justin

Nathanael Jo

Andrés Gómez

Phebe Vayanos

PVOSSLER@USC.EDU

SAGHAEI@USC.EDU

NJUSTIN@USC.EDU

NATHANAEL.JO@GMAIL.COM

GOMEZAND@USC.EDU

PHEBE.VAYANOS@USC.EDU

*University of Southern California, Center for AI in Society, Los Angeles, CA 90089*

## Abstract

ODTlearn is an open-source Python package that provides methods for learning optimal decision trees for high-stakes predictive and prescriptive tasks based on the mixed-integer optimization (MIO) framework proposed in (Aghaei et al., 2021) and several of its extensions. The current version of the package provides implementations for learning optimal classification trees, optimal fair classification trees, optimal classification trees robust to distribution shifts, and optimal prescriptive trees from observational data. We have designed the package to be easy to maintain and extend as new optimal decision tree problem classes, reformulation strategies, and solution algorithms are introduced. To this end, the package follows object-oriented design principles and supports both commercial (Gurobi) and open source (COIN-OR branch and cut) solvers. The package documentation and an extensive user guide can be found at <https://d3m-research-group.github.io/odtlearn/>. Additionally, users can view the package source code and submit feature requests and bug reports by visiting <https://github.com/D3M-Research-Group/odtlearn>.

**Keywords:** Mixed-integer optimization, prescriptive trees, classification trees, distribution shifts, fair classification trees, robust classification trees, open source software.

## 1. Introduction

Automated data-driven predictive and prescriptive methods are increasingly being used in high-stakes domains to inform and support decision-making. In such settings, these tools should be: (a) *accurate* (to minimize erroneous predictions/prescriptions that may negatively affect the populations on which they are deployed), (b) *interpretable* (so that predictions and decisions are transparent, accountable, and easy to audit), (c) *flexible* (i.e., possible to easily augment with domain specific constraints such as capacity and/or fairness constraints), and (d) *robust* (to ensure high-quality solutions even under adversarial shifts between training and deployment data).

Despite their popularity, decision trees (Breiman et al., 1984) are not necessarily well suited for data-driven decision-making in high-stakes domains. While the structure of decision trees makes them easy to interpret, they are typically constructed using heuristics and may yield suboptimal solutions. Furthermore, with heuristic-based decision trees, it is

---

\*. Corresponding Author

not immediately apparent how to incorporate relevant side information into the tree construction process or how to make a tree robust to adversarial shifts between training and deployment data. Optimal decision trees retain the interpretability of heuristic decision trees while providing the flexibility to model the types of problems decision-makers face and deliver optimal, trustworthy solutions.

Our **ODTlearn** Python package provides methods for fitting provably optimal decision trees using mixed-integer optimization (MIO) for various problem types and settings commonly encountered by practitioners in high-stakes settings. **ODTlearn** implements the FlowOCT modeling and solution paradigm proposed by Aghaei et al. (2021), which is significantly faster and provides better out-of-sample performance than previous MIO-based algorithms (see Section 2 for additional discussion). This approach generalizes beyond standard classification problems to problems involving imbalanced datasets (e.g., by optimizing weighted accuracy or worst-case accuracy, by constraining recall or precision, or by balancing sensitivity and specificity). In addition to the core algorithm, we implement its generalization to learn optimal fair decision trees (FairOCT) that optimize accuracy while satisfying arbitrary domain specific fairness constraints such as statistical parity, conditional statistical parity, or equalized odds as proposed in (Jo et al., 2022). Experiments demonstrated that these methods outperform popular heuristic-based algorithms for learning fair trees. For users deploying decision trees in settings with potential distribution shifts between training and testing, we implement the MIP formulation proposed in (Justin et al., 2021) (RobustOCT) for learning optimal robust classification trees. Finally, we implement the MIP formulation proposed in Jo et al. (2022) (FlowOPT) for learning optimal prescriptive trees from observational data. This framework can be used to design treatment assignment policies in the form of decision trees, being highly interpretable while offering a tunable degree of personalization. Importantly, the learned trees can also be constrained to satisfy domain specific requirements such as budget constraints (e.g., limited amount of treatments) or fairness constraints (e.g., conditional statistical parity in allocation).

The remainder of this paper is organized as follows. Section 2 compares **ODTlearn** to related packages. We discuss the structure of the package in Section 3. We explain how our package interacts with MIP solvers and provide a usage example in Section 4. Finally, the quality practices under which the package is developed are described in Section 5.

## 2. Comparison to Related Software

There are numerous high-quality implementations of heuristic-based methods for learning decision trees in R and Python, including **scikit-learn** (Pedregosa et al., 2011), **Rpart** (Therneau et al., 1997), and **Caret** (Kuhn et al., 2008) among many others. As discussed in Section 1, these methods are fast, but lack the modeling flexibility and optimality guarantees of MIO-based methods.

Several different R and Python packages provide implementations of methods for constructing optimal decision trees using approaches other than mixed-integer optimization. The **evtree** (Grubinger et al., 2014) Python package implements an evolutionary algorithm that searches the space of possible decision trees to find globally optimal decision trees. **PyDL8.5** (Aglin et al., 2021) is an open-source Python package that implements the DL8.5 (Aglin et al., 2020) algorithm in C++ for learning optimal decision trees through the

use of itemset mining techniques. The `gosdt` Python package<sup>1</sup> provides a C++ implementation of Generalized Optimal Sparse Decision Trees (GOSDT) (Lin et al., 2020; McTavish et al., 2022), which employs a branch-and-bound method with dynamic programming, similar to DL8.5. GOSDT can directly model continuous features and construct decision trees for various objectives, including accuracy, AUC, and partial area under ROC convex hull. While the MIO formulations used in `ODTlearn` provide the flexibility to incorporate side constraints, they are generally slower than the methods used in `evtree`, DL8.5, and GOSDT, especially for larger datasets or deeper trees. These methods employ specialized algorithms, such as evolutionary algorithms or branch-and-bound with dynamic programming, which can often find optimal trees more quickly than general-purpose MIO solvers. However, it is not immediately obvious how to incorporate side constraints into the tree construction process for these methods, which is a strength of the MIO approach.

The `policytree` package (Sverdrup et al., 2020) provides an R implementation of Policy Trees (PT) (Zhou et al., 2022), which learn personalized treatment assignment policies from observational data using decision trees and doubly robust estimators to handle confounding. PT uses a recursive partitioning approach, allowing it to learn trees that handle continuous and discrete covariates without pre-discretization and in less time than the FlowOPT MIO formulation. However, PT’s recursive partitioning approach may not converge to the global optimum and cannot easily incorporate complex constraints.

There are also open-source implementations of other MIO formulations for learning optimal classification trees. The authors of the BinOCT MIO formulation (Verwer and Zhang, 2019) provide an open-source implementation<sup>2</sup> of their method. Additionally, Tang (2021) provides open-source implementations of the BinOCT, OCT (Bertsimas and Dunn, 2017), and FlowOCT (Aghaei et al., 2021) MIO formulations. While both projects provide valuable implementations of these methods for technical users and researchers, these projects are less accessible to a general audience with limited knowledge of MIO. Neither project distributes its code as a Python package, and they are not indexed on PyPi. Furthermore, their implementations rely exclusively upon proprietary MIP solvers such as CPLEX or Gurobi, which require paid or academic licenses to use with large-scale problems.

In comparisons on simulated data, the FlowOCT and FlowOPT formulations solve problem instances faster and are more accurate out-of-sample compared to existing MIO formulations. In Aghaei et al. (2021), the authors implement FlowOCT, BinOCT, and OCT in Gurobi to directly compare the in-sample and out-of-sample performance of the methods. When comparing the speed at which each method solves a fixed number of problem instances, they find that FlowOCT is 29 times faster than the other two formulations at solving problems with balanced decision trees. This significant speedup is attributed to the stronger linear optimization relaxation of the FlowOCT formulation, which leads to more effective pruning of the branch-and-bound tree during the solving process. Regarding out-of-sample accuracy, FlowOCT improves accuracy compared to BinOCT and OCT by up to 7% and 21%, respectively. In Jo et al. (2021), the authors compare FlowOPT to Gurobi implementations of the MIO formulations proposed in Kallus (2017) (K-PT) and Bertsimas et al. (2019) (B-PT). FlowOPT outperforms K-PT and B-PT in terms of the out-of-sample probability of correct treatment assignment across various experiment designs.

---

1. <https://pypi.org/project/gosdt/>

2. <https://github.com/SiccoVerwer/binoct>

Interpretable AI (Interpretable AI, 2022) offers a proprietary Julia (Bezanson et al., 2017) package with a Python wrapper that provides methods for learning optimal classification, prescriptive, and regression trees. These implementations use the heuristic algorithm based on local search proposed by Bertsimas and Dunn (2019) (LST). On the one hand, LST is much faster than MIO formulations like FlowOCT, requiring only seconds to find a local optimum. On the other hand, MIO formulations are more flexible and able to solve instances to provable optimality.

In summary, **ODTlearn** distinguishes itself from existing implementations of optimal decision tree methods in several key ways. First, it provides open-source, user-friendly implementations of state-of-the-art MIO formulations for learning optimal decision trees, including FlowOCT and FlowOPT, which have been shown to outperform other MIO formulations in terms of solution speed and out-of-sample performance. Second, **ODTlearn** is distributed as a Python package indexed on PyPi, making it easily accessible to a broad audience. Third, **ODTlearn** supports open-source and proprietary MIP solvers, allowing users without access to commercial solvers to still benefit from the package. Finally, the MIO formulations in **ODTlearn** provide the flexibility to incorporate custom constraints and objectives, enabling users to tailor the learned decision trees to specific requirements of the problem they wish to solve.

### 3. Software Architecture

The software architecture of **ODTlearn** is motivated by the optimal decision tree literature in which researchers have recently proposed numerous new problem classes, reformulation strategies, and solution approaches. With this in mind, we have created a class structure for our package, summarized in Figure 1, that follows the SOLID principles of object-oriented programming for developing software (Martin, 2003). These principles emphasize structuring classes that are easy to maintain and extend. The Single Responsibility Principle (SRP) states that each class should have a single responsibility or purpose. The Open-Closed Principle (OCP) requires that users should be able to extend a class’s behavior without modifying its existing code. According to the Liskov Substitution Principle (LSP), derived classes should be able to replace their base classes without affecting correctness. The Interface Segregation Principle (ISP) asserts that users should not be forced to depend on methods they do not use. Finally, the Dependency Inversion Principle (DIP) suggests that software design should rely upon abstract interfaces rather than concrete implementations. These principles establish practices for developing software with considerations for maintaining and extending it as the project grows.

Following the dependency inversion principle, all classes in **ODTlearn** are derived from our abstract base class **OptimalDecisionTree**. This class provides a standardized interface for the two types of trees currently supported (classification and prescription) while keeping their implementation details separate to follow the interface segregation principle. The **OptimalDecisionTree** abstract class allows our class structure to follow the open-closed principle as we can create the derived classes **OptimalClassificationTree** and **OptimalPrescriptiveTree** with problem-specific methods for traversing and visualizing the decision tree. Next, the children of **OptimalClassificationTree** and **OptimalPrescriptiveTree** implement methods for creating the decision variables, the constraints, and

the objective function necessary for constructing the optimization problem of interest. The separate classes for each of the variations of the MIO formulations ensure that each of the derived classes has a single job, obeying the single responsibility principle. Finally, the classes in the third and fourth levels of Figure 1 implement user-facing methods such as `fit` and `predict` required by their parent classes. This ensures that our low-level classes respect the Liskov substitution principle by adhering to the basic requirements of their parent class. Thus, our adherence to the SOLID principles ensures that researchers and practitioners building upon `ODTlearn` can easily augment it with more features (e.g., different objectives or additional constraints) or even build new types of trees.

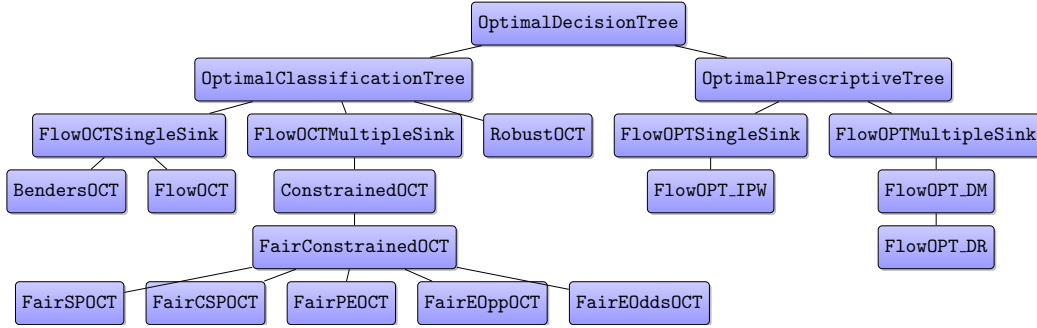


Figure 1: Inheritance Diagram for `ODTlearn`.

## 4. Implementation Details

`ODTlearn` is built upon the Python MIP package (Santos and Toffolo, 2020), which provides a unified interface for implementing MIO formulations with callbacks across both open-source and commercial solvers. By leveraging Python MIP’s consistent interface, our package is able to support both the open-source CBC solver and the commercial Gurobi solver without the need for solver-specific code. This design choice allows us to focus on implementing the core functionality of `ODTlearn` while still supporting a wide range of users with different solver preferences and access to commercial licenses. The use of Python MIP is particularly important for our `BendersOCT` formulation, which relies on callbacks to implement the Benders’ decomposition algorithm. Callbacks are essential for the efficient solution of large-scale problems using `BendersOCT`, and Python MIP’s interface allows us to implement these callbacks in a solver-independent manner. Furthermore Python MIP interacts directly with the solver’s native dynamic loadable library (DLL) interface, ensuring that any computationally intensive steps are performed by the solver and not in Python.

For fitting optimal classification and prescriptive trees, we provide users with an interface that follows a common fit-predict structure. Figure 2 provides a code snippet demonstrating how to use the `ODTlearn` API to fit a fair optimal classification tree. Once an optimal decision tree has been learned, users can employ the build in `plot_tree` function to visualize the tree.

```

from odttlearn.datasets import fairness_example
from odttlearn.fair_oct import FairSPOCT
import matplotlib.pyplot as plt

X, y, protect_feat, legit_factor = fairness_example()
fcl = FairSPOCT(solver="gurobi", positive_class=1, depth=2, _lambda=0.01, fairness_bound=1)
fcl.fit(X, y, protect_feat, legit_factor)
fcl_predictions = fcl.predict(X)

fcl.plot_tree()
plt.show()

```

Figure 2: Code for fitting an optimal fair classification tree on a toy data set.

## 5. Development

Releases of the ODTlearn package are available via PyPI at <https://pypi.org/project/odttlearn>. The package source code and documentation are hosted on GitHub (<https://github.com/D3M-Research-Group/odttlearn>). Collaboration in the form of discussions, feature requests, or bug reports is made possible through the GitHub issue and pull request workflow. We have implemented continuous integration through GitHub Actions to ensure backward compatibility and quickly identify any code regressions. Our documentation includes installation instructions, a user guide, an API reference, and downloadable example notebooks demonstrating each of the classification methods implemented in the package. The documentation is hosted via GitHub pages at <https://d3m-research-group.github.io/odttlearn>. The package is distributed under the GPL-3.0 license and makes use of several core libraries within Python’s scientific computing ecosystem: `scikit-learn` (Pedregosa et al., 2011), `numpy` (Harris et al., 2020), and `pandas` (McKinney et al., 2010).

## Acknowledgments

P. Vayanos and N. Justin are funded in part by the National Science Foundation under CAREER award number 2046230. P. Vayanos and P. Vossler acknowledge support from the USC Zumberge Special Solicitation – Epidemic & Virus Related Research and Development award. N. Jo acknowledges support from the Epstein Institute at the University of Southern California. A. Gómez is funded in part by the National Science Foundation under grant 2152777. N. Justin is funded in part by the National Science Foundation under the Graduate Research Fellowship Program. P. Vayanos and A. Gómez are funded in part by the National Science Foundation under NRT grant 2346058. They are grateful for the support.

## References

- Sina Aghaei, Andrés Gómez, and Phebe Vayanos. Strong optimal classification trees. *arXiv preprint arXiv:2103.15965*, 2021.
- Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3146–3153, 2020.
- Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Pydl8. 5: a library for learning optimal decision trees. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 5222–5224, 2021.
- Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7): 1039–1082, 2017.
- Dimitris Bertsimas and Jack Dunn. *Machine learning under a modern optimization lens*. Dynamic Ideas LLC Charlestown, MA, 2019.
- Dimitris Bertsimas, Jack Dunn, and Nishanth Mundru. Optimal prescriptive trees. *INFORMS Journal on Optimization*, 1(2):164–183, 2019.
- Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi: 10.1137/141000671. URL <https://epubs.siam.org/doi/10.1137/141000671>.
- Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 1984.
- Thomas Grubinger, Achim Zeileis, and Karl-Peter Pfeiffer. Evtree: Evolutionary learning of globally optimal classification and regression trees in R. *J. Stat. Softw.*, 61(1):1–29, October 2014.
- Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with Numpy. *Nature*, 585(7825):357–362, 2020.
- LLC Interpretable AI. Interpretable AI documentation, 2022. URL <https://www.interpretable.ai>.
- Nathanael Jo, Sina Aghaei, Andrés Gómez, and Phebe Vayanos. Learning optimal prescriptive trees from observational data. *arXiv preprint arXiv:2108.13628*, 2021.
- Nathanael Jo, Sina Aghaei, Jack Benson, Andrés Gómez, and Phebe Vayanos. Learning optimal fair classification trees. *arXiv preprint arXiv:2201.09932*, 2022.
- Nathan Justin, Sina Aghaei, Andres Gomez, and Phebe Vayanos. Optimal robust classification trees. In *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond*, 2021.

- Nathan Kallus. Recursive partitioning for personalization using observational data. In *International conference on machine learning*, pages 1789–1798. PMLR, 2017.
- Max Kuhn et al. Caret package. *Journal of statistical software*, 28(5):1–26, 2008.
- Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. *arXiv [cs.LG]*, June 2020.
- Robert Cecil Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.
- Wes McKinney et al. Data structures for statistical computing in Python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.
- Hayden McTavish, Chudi Zhong, Reto Achermann, Ilias Karimalis, Jacques Chen, Cynthia Rudin, and Margo Seltzer. Fast sparse decision tree optimization via reference ensembles. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pages 9604–9613, 2022.
- Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- Haroldo G Santos and T Toffolo. Mixed integer linear programming with Python. *Accessed: Apr*, 2020.
- Erik Sverdrup, Ayush Kanodia, Zhengyuan Zhou, Susan Athey, and Stefan Wager. policytree: Policy learning via doubly robust empirical welfare maximization over trees. *J. Open Source Softw.*, 5(50):2232, June 2020.
- Bo Tang. Optimal classification tree. [https://github.com/LucasBoTang/Optimal\\_Classification\\_Trees](https://github.com/LucasBoTang/Optimal_Classification_Trees), 2021.
- Terry M Therneau, Elizabeth J Atkinson, et al. An introduction to recursive partitioning using the rpart routines. Technical report, Technical report Mayo Foundation, 1997.
- 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.
- Zhengyuan Zhou, Susan Athey, and Stefan Wager. Offline multi-action policy learning: Generalization and optimization. *Operations Research*, 2022.