Optimal Trees for Prediction and Prescription

by

Jack William Dunn

B.E.(Hons), University of Auckland (2014)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author ....
Sloan School of Management
May 18, 2018

Certified by..
Dimitris Bertsimas
Boeing Professor of Operations Research
Co-director, Operations Research Center
Thesis Supervisor

Accepted by ....
Patrick Jaillet
Dugald C. Jackson Professor
Department of Electrical Engineering and Computer Science
Co-Director, Operations Research Center

Optimal Trees for Prediction and Prescription

by

Jack William Dunn

Submitted to the Sloan School of Management
on May 18, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

For the past 30 years, decision tree methods have been one of the most widely-used approaches in machine learning across industry and academia, due in large part to their interpretability. However, this interpretability comes at a price—the performance of classical decision tree methods is typically not competitive with state-of-the-art methods like random forests and gradient boosted trees.

A key limitation of classical decision tree methods is their use of a greedy heuristic for training. The tree is therefore constructed one locally-optimal split at a time, and so the final tree as a whole may be far from global optimality. Motivated by the increase in performance of mixed-integer optimization methods over the last 30 years, we formulate the problem of constructing the optimal decision tree using discrete optimization, allowing us to construct the entire decision tree in a single step and hence find the single tree that best minimizes the training error. We develop high-performance local search methods that allow us to efficiently solve this problem and find optimal decision trees with both parallel (axes-aligned) and hyperplane splits.

We show that our approach using modern optimization results in decision trees that improve significantly upon classical decision tree methods. In particular, across a suite of synthetic and real-world classification and regression examples, our methods perform similarly to random forests and boosted trees whilst maintaining the interpretability advantage of a single decision tree, thus alleviating the need to choose between performance and interpretability.

We also adapt our approach to the problem of prescription, where the goal is to make optimal prescriptions for each observation. While constructing the tree, our method simultaneously infers the unknown counterfactuals in the data and learns to make optimal prescriptions. This results in a decision tree that optimizes both the predictive and prescriptive error, and delivers an interpretable solution that offers significant improvements upon the existing state-of-the-art in prescriptive problems.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Professor of Operations Research
Co-director, Operations Research Center

Acknowledgments

Firstly, I would like to thank Dimitris Bertsimas for being not just an advisor, but also a mentor and true friend to me during my time at the ORC. I am constantly inspired by his visions, ambitions, and positivity, and it is undeniable that he has had a great and positive influence on my attitude to both research and life. I am continually impressed by his aptitude for selecting practically important problems—he asked me to think about optimizing decision trees during our first meeting, and I never would have imagined that we would still be meeting about them nearly four years later! It has been a truly rewarding experience to work with and be mentored by someone with such a genuine drive to pursue and address with research the things that matter most in life.

I would like to thank Rahul Mazumder and Nikos Trichakis for serving as the other two members of my thesis committee, and for their useful feedback and insights during my thesis proposal. Thanks also to Juan Pablo Vielma for his role on my general examination committee.

I am extremely grateful to all of my collaborators during my time at the ORC, including the many collaborations that were not included in the thesis. I would especially like to thank the following people: Colin Pawlowski and Daisy Zhuo for our work in the early years on Robust Classification, a project we never seemed to see the end of; Yuchen Wang for his collaborations on a variety of projects and being one of the most prolific users of Optimal Trees; Lauren Berk for her assistance on our project with Massachusetts General Hospital; Nishanth Mundru for his creativity in our work in figuring out Optimal Prescriptive Trees; and Emma Gibson and Agni Orfanoudaki for their perseverance with Optimal Survival Trees. It has been an absolute pleasure working with so many talented people over the course of my PhD and I hope it can continue into the future. I also thank rest of the wider ORC community for making the experience an enjoyable one.

My thanks also goes to the various outside collaborators I have worked with. George Velmahos and Haytham Kaafarani at Massachusetts General Hospital were both great inspirations and taught me a lot about the realities of working in healthcare. Tom Trikalinos provided a lot of valuable input in our work with diagnosing head trauma in children. Aris Paschalidis, Steffie van Loenhout and Korina Digalaki all spent summers working with me in various capacities, and I am thankful for their assistance and company.

One of the most demanding and yet rewarding aspects of the PhD has been developing and supporting the use of the Optimal Trees package within the ORC and beyond. I am extremely grateful to the roughly 30 students that have been using Optimal Trees in their own research, both for keeping me busy with bug reports and feature requests, but also providing the impetus and drive for continued improvement of the software. I would also like to thank Daisy Zhuo specifically for her contributions to the package, and for keeping me passionate and excited about the development process with her extreme enthusiasm for life.

Finally, I would like to acknowledge and thank my family for their unconditional support and encouragement both during my time here and over my whole life. I cannot thank you enough for the assistance and direction you have given me, and for your impact in making me who I am today. Last, but certainly not least, I would like to thank my partner Jenny, who despite the inherent challenges has managed to follow me over the pond to the US and continues to be my biggest supporter every day.

Contents

List of Figures **11**

List of Tables **15**

1 Introduction **17**

1.1 Overview of CART 18

1.2 Limitations of CART 23

1.3 The Remarkable Progress of MIO 26

1.4 What is Tractable? 28

1.5 Motivation, Philosophy and Goal 29

1.6 Structure of the Thesis 31

2 Optimal Classification Trees with Parallel Splits **33**

2.1 Review of Classification Tree Methods 34

2.2 A Mixed-Integer Optimization Approach 38

2.3 A Local Search Approach 47

2.4 A Method for Tuning Hyperparameters 57

2.5 Experiments with Synthetic Datasets 67

2.6 Experiments with Real-World Datasets 75

2.7 Conclusions 83

3 Optimal Classification Trees with Hyperplane Splits **85**

3.1 OCT with Hyperplanes via MIO 86

3.2 OCT with Hyperplanes via Local Search 90

3.3 The Accuracy-Interpretability Tradeoff 98
3.4 Experiments with Synthetic Datasets 101
3.5 Experiments with Real-World Datasets 110
3.6 Conclusions 114

4 Optimal Regression Trees with Constant Predictions 117

4.1 Review of Regression Tree Methods 118
4.2 ORT with Constant Predictions via MIO 119
4.3 ORT with Constant Predictions via Local Search 123
4.4 Experiments with Synthetic Datasets 126
4.5 Experiments with Real-World Datasets 139
4.6 Conclusions 145

5 Optimal Regression Trees with Linear Predictions 147

5.1 ORT with Linear Predictions via MIO 149
5.2 ORT with Linear Predictions via Local Search 152
5.3 Experiments with Synthetic Datasets 155
5.4 Experiments with Real-World Datasets 162
5.5 Conclusions 166

6 Optimal Prescription Trees 169

6.1 Optimal Prescriptive Trees 177
6.2 Experiments with Synthetic Datasets 181
6.3 Experiments with Real-World Datasets 190
6.4 Conclusions 200

7 Diagnostics for Children After Head Trauma 201

7.1 Data and Methods 202
7.2 Results 205
7.3 Discussion 212

8 Conclusion 215

List of Figures

1-1 An example of the CART algorithm applied to the “Iris” dataset that shows the recursive partitioning step-by-step. 20

1-2 The tree learned by CART for the “Iris” dataset. 21

1-3 The CART tree for the “Iris” dataset after conducting pruning. 22

1-4 An example of partitioning the “Iris” dataset with arbitrary hyperplanes rather than just splits parallel to the axes. 23

1-5 A tree for the “Iris” dataset with hyperplane splits. 23

1-6 An example where greedy tree induction fails to learn the truth in the data. 24

1-7 Peak supercomputer speed in GFlop/s (log scale) from 1994–2016. . . 28

2-1 An example of a decision tree. 35

2-2 An example of a strong split being hidden behind a weak split. 37

2-3 The maximal tree for depth $D = 2$. 39

2-4 Comparison of upper and lower bound evolution while solving MIO problem (2.24). 46

2-5 Training accuracy (%) and running time for each method on “Banknote Authentication” dataset. 49

2-6 Training accuracy (%) and running time for each method on “Banknote Authentication” dataset. 57

2-7 Example of pruning process following Algorithm 2.4 on the “Breast Cancer Wisconsin” dataset. 61

2-8 An example showing the instability of cost-complexity pruning. . . . 63

2-9 Tuning results using batch pruning on "Breast Cancer Wisconsin" dataset. 65
2-10 Comparison of pruning methods on "Breast Cancer Wisconsin" dataset in the region where validation error is minimized. 66
2-11 Synthetic experiments showing the effect of training set size. 69
2-12 Synthetic experiments showing the effect of number of features. . . . 70
2-13 Synthetic experiments showing the effect of number of classes. 71
2-14 Synthetic experiments showing the effect of number of random restarts. 72
2-15 Synthetic experiments showing the effect of feature noise. 73
2-16 Synthetic experiments showing the effect of label noise. 74
2-17 Mean out-of-sample accuracy for each method across all 60 datasets. . 78
2-18 A comparison of CART and OCT trees on the "Banknote Authentication" dataset. 80
2-19 Improvement in out-of-sample accuracy for OCT over CART for each of the 60 datasets, according to ratio of n and $\log(p)$. 82

3-1 Training accuracy (%) and running time for each method on "Blood Transfusion Service Center" dataset. 91
3-2 Training accuracy (%) and running time for each method on "Blood Transfusion Service Center" dataset. 96
3-3 Comparison of optimal trees with parallel and hyperplane splits on the "Banknote Authentication" dataset. 100
3-4 Synthetic experiments showing the effect of training set size. 102
3-5 Synthetic experiments showing the effect of number of features. . . . 103
3-6 Synthetic experiments showing the effect of number of classes. 104
3-7 Synthetic experiments showing the effect of number of random restarts. 104
3-8 Synthetic experiments showing the effect of feature noise. 105
3-9 Synthetic experiments showing the effect of label noise. 106
3-10 Synthetic experiments showing the effect of split density. 108
3-11 Synthetic experiments showing the effect of number of random hyperplane restarts. 109

3-12 Mean out-of-sample accuracy for each method across all 60 datasets. . 111

4-1 CART regression tree predicting fuel consumption of automobiles using the “mtcars” dataset. 119
4-2 Training mean squared error and running time for each method on “Hybrid Vehicle Prices” dataset. 124
4-3 Training mean squared error and running time for each method on “Hybrid Vehicle Prices” dataset. 126
4-4 Synthetic experiments showing the effect of training set size. 127
4-5 Synthetic experiments showing the effect of number of features. . . . 128
4-6 Synthetic experiments showing the effect of number of random restarts. 129
4-7 Synthetic experiments showing the effect of feature noise. 130
4-8 Synthetic experiments showing the effect of label noise. 131
4-9 Synthetic experiments showing the effect of training set size. 132
4-10 Synthetic experiments showing the effect of number of features. . . . 133
4-11 Synthetic experiments showing the effect of number of random restarts. 134
4-12 Synthetic experiments showing the effect of feature noise. 135
4-13 Synthetic experiments showing the effect of label noise. 136
4-14 Synthetic experiments showing the effect of split density. 137
4-15 Synthetic experiments showing the effect of number of random hyperplane restarts. 138
4-16 Mean out-of-sample R^2 for each method across all 26 datasets. 141
4-17 Mean out-of-sample R^2 for each method across all 26 datasets. 143

5-1 Synthetic experiments showing the effect of training set size. 156
5-2 Synthetic experiments showing the effect of number of features. . . . 156
5-3 Synthetic experiments showing the effect of number of random restarts. 157
5-4 Synthetic experiments showing the effect of feature noise. 157
5-5 Synthetic experiments showing the effect of label noise. 158
5-6 Synthetic experiments showing the effect of regression density. 159
5-7 Synthetic experiments showing the effect of split density. 160

5-8 Synthetic experiments showing the effect of regression density. 161
5-9 Mean out-of-sample R^2 for each method across all 26 datasets. 164

6-1 Test prediction and personalization error as a function of μ 178
6-2 Performance results for Experiment 1. 185
6-3 Tree constructed by OPT(0.5)-L for an instance of Experiment 1. . . 186
6-4 Performance results for Experiment 2. 186
6-5 Tree constructed by OPT(0.5)-L for an instance of Experiment 2. . . 188
6-6 Performance results for Experiment 3. 189
6-7 Misclassification rate for warfarin dosing prescriptions as a function of training set size. 193
6-8 Comparison of methods for personalized diabetes management. The leftmost plot shows the overall mean change in HbA1c across all patients (lower is better). The center plot shows the mean change in HbA1c across only those patients whose prescription differed from the standard-of-care. The rightmost plot shows the proportion of patients whose prescription was changed from the standard-of-care. 196
6-9 Out-of-sample average personalized income as a function of inclusion rate. 198

7-1 PECARN rules for ciTBI in children younger than 2 years. 206
7-2 OCT for ciTBI in children younger than 2 years. 207
7-3 PECARN rules for ciTBI in children at least 2 years old. 208
7-4 OCT for ciTBI in children at least 2 years old. 208

List of Tables

2.1 Full results for CART and OCT at depth 10. 77

2.2 Mean out-of-sample accuracy (%) results for CART and OCT across all 60 datasets. 79

2.3 Number of datasets where CART and OCT were strongest across all 60 datasets. 81

3.1 Performance results for classification methods (with maximum depth 10) on each of the 60 datasets. 113

3.2 Pairwise significance tests for performance of classification methods (with maximum depth 10) across 60 datasets. 114

4.1 Full results for CART and ORT at depth 10. 140

4.2 Mean out-of-sample R^2 results for CART and ORT across all 26 datasets. 142

4.3 Performance results for regression methods (with maximum depth 10) on each of the 26 datasets. 144

4.4 Pairwise significance tests for performance of regression methods (with maximum depth 10) across 26 datasets. 145

5.1 Performance results for regression methods (with maximum depth 10) on each of the 26 datasets. 163

5.2 Pairwise significance tests for performance of regression methods (with maximum depth 10) across 26 datasets. 165

6.1 Average personalized income on the test set for various methods, arranged in increasing order. 198
6.2 Average R^2 on the test set for each method when estimating the personalized treatment effect. 200

7.1 Prospectively collected predictors in the PECARN cohort. 204
7.2 Cross-classification of predictions of higher and low risk and ciTBI status in children younger than 2 years. 210
7.3 Cross-classification of predictions of higher and low risk and ciTBI status in children 2 years and older. 210
7.4 Patients with ciTBI who were missed by the PECARN or OCT rules. 210
7.5 Predictive performance of OCT versus PECARN rules among children younger than 2 years. 211
7.6 Predictive performance of OCT versus PECARN rules among children at least 2 years old. 211

Chapter 1

Introduction

Machine Learning methods represent one of the most promising and widely applicable methodologies of our times that aspires to tackle some of the world's most important questions:

- What is intelligence?
- Can we cure some of the most significant human diseases like cancer and cardiovascular diseases using electronic medical records and genomic data?
- Can we improve human productivity, for example by creating self-driving cars?

Decision trees are one of the most widely-used techniques in Machine Learning used for the central problems of regression and classification. The leading work for decision tree methods in classification is CART (Classification and Regression Trees), proposed by Breiman et al. [25]. This book has roughly 37,000 Google Scholar citations (as of February 2018) and it is one of the most cited works in the mathematical sciences. Guided by the training data $(\mathbf{x}_i, y_i), i = 1, \dots n$, decision trees recursively partition the feature space for classification problems and assign a label to each resulting partition. The space is partitioned into boxes using splits parallel to the coordinate axes: Is $x_i < a$ or $x_i \geq a$? The tree is then used to classify future points according to these splits and labels. The key advantage of decision trees over other methods is that they are very interpretable, and in many applications this interpretability is often preferred over other less-interpretable methods with higher accuracy. In healthcare for instance, decision trees resemble a doctor's thinking in that doctors typically evaluate a patient's medical condition one variable at a time, and thus are very useful for assisting doctors in their work. In contrast, other methods that are not consistent with human thinking are far less useful in this setting.

In this chapter, we give an overview of the CART algorithm, outline some of the limitations of CART, discuss the remarkable progress of discrete optimization which is a key tool we use in the thesis, and outline the motivation, philosophy, objectives and structure of the thesis.

1.1 Overview of CART

CART takes a top-down approach to determining the partitions in the tree. Starting from the root node, a split is determined by solving an optimization problem to find the best split, before dividing the points according to the split and recursing on the two resulting child nodes. CART chooses the best split using an impurity measure, which is a non-linear quantity that measures the similarity of labels among points in a group. CART seeks the split that divides the points into two groups such that the sum of the impurities of each group is minimized. Typical choices for the impurity measure are the *gini* or *twoing* criteria for classification problems, and the *mean squared error* or *mean absolute error* for regression.

This greedy, recursive partitioning process continues at each new node until one of the following stopping criteria is met:

- The node being partitioned has fewer points than the minimum allowed leaf size, a parameter of CART denoted $N_{\min}$;
- The impurity of the node cannot be reduced by any candidate split, e.g., if all points in the node have the same label.

When the partitioning has concluded, each leaf node in the tree is assigned a label that is used for predicting the labels of new data points. The label for the leaf is assigned using the labels of the training points that fall into this leaf node; typically the mode (most common) of these labels is used for classification problems, and the mean of these labels is used for regression problems.

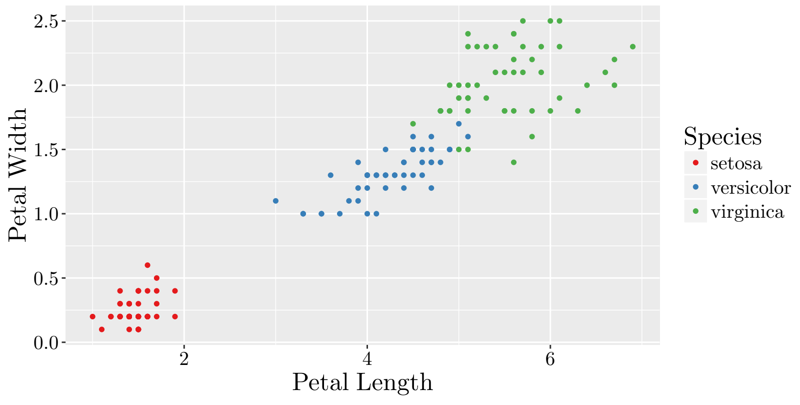
To illustrate this process, in Figure 1-1 we provide an example of the CART algorithm applied to Fisher's "Iris" dataset, a classic and well-known classification dataset from the UCI Machine Learning Repository [77]. Each of the 150 points in the dataset corresponds to an iris flower which is one of three species ("setosa", "versicolor" or "virginica"). There are four measurements recorded for each flower: the length and width of the sepals and petals. The goal is to predict the species of iris using only the four physical measurements. In Figure 1-1a, we plot the data plotted according to the petal length and width. Figures 1-1b–1-1e show the progression of the CART algorithm on this data. CART first splits on the petal length, perfectly separating the "setosa" species from the others, and so no further splitting is needed on the left side of this split. The second split is on the petal width, and largely separates the "versicolor" and "virginica" species from each other, although the separation is not perfect. The third and fourth splits refine this separation so that all partitions are pure. Figure 1-1f shows the final labels assigned to each partition of the space, which correspond to the leaf nodes on the CART tree. The final partitioning is also shown in tree form in Figure 1-2.

One of the primary concerns when growing the tree is to avoid overfitting the tree to the training data. It is possible to achieve very high accuracy on the training points with a large tree, but such a tree is then likely to perform poorly when making predictions on new data. We can restrict the degree of overfitting by controlling the tradeoff between the training accuracy and the number of splits in the tree, which is also known as the *complexity* of the tree.

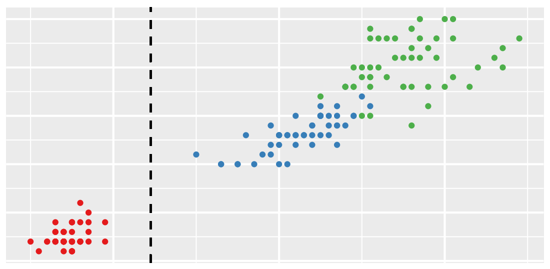
CART controls this tradeoff between accuracy and complexity by introducing a penalty on the complexity of the tree. Each split in the tree is required to improve the accuracy by more than the value of the *complexity parameter*, denoted α, otherwise the split will not be included in the tree. By adjusting the value of α, we can change the complexity of the final tree and thus control the degree of overfitting to the

Figure 1-1: An example of the CART algorithm applied to the “Iris” dataset that shows the recursive partitioning step-by-step.

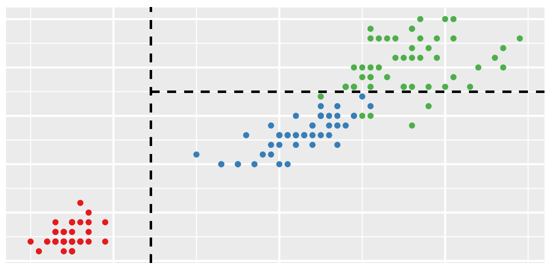
(a) Data



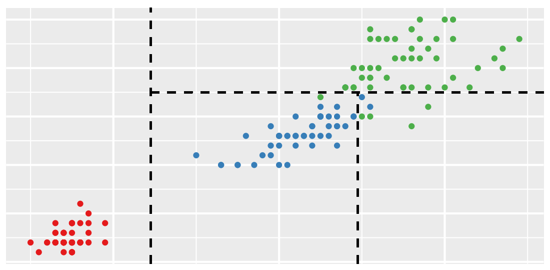
(b) First split



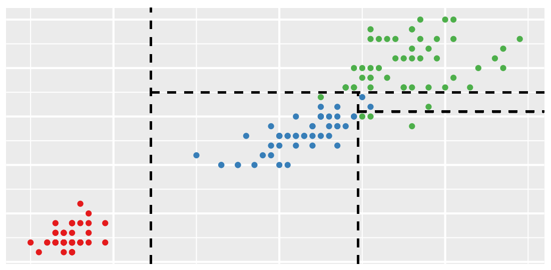
(c) Second split



(d) Third split



(e) Fourth split



(f) Final predictions

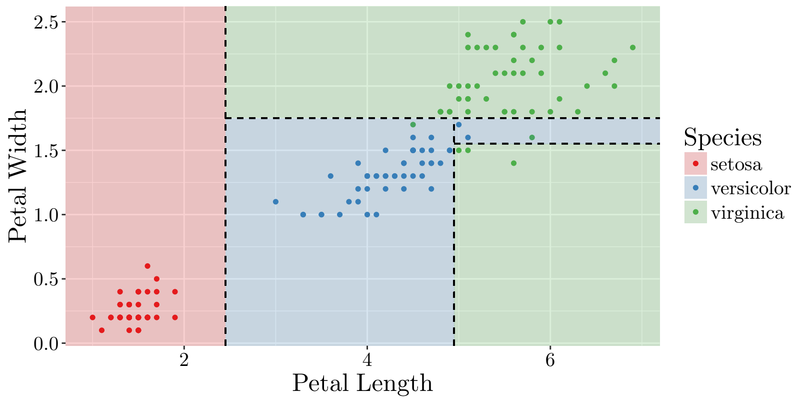
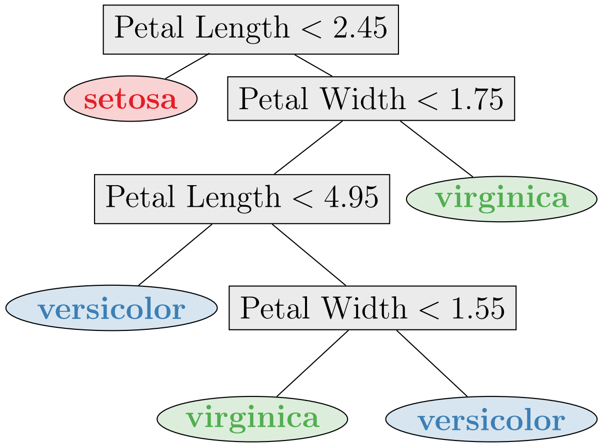


Figure 1-2: The tree learned by CART for the "Iris" dataset.

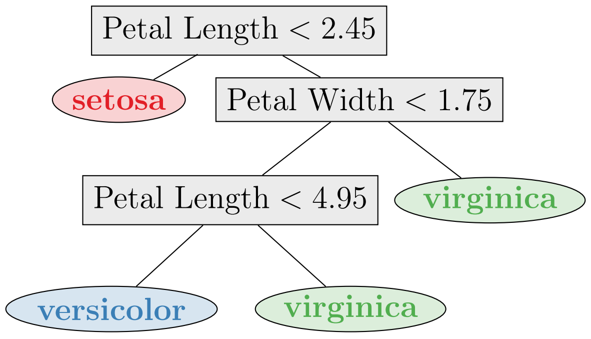


training data. The penalty on complexity is applied retroactively after the tree has been grown. At each split in the tree, we consider replacing the split and its children with a single leaf instead, and calculate the change in accuracy resulting from this change. If the accuracy improvement due to the split is lower than α, we replace this split with a leaf. This process is repeated until all splits in the tree have an accuracy improvement of at least α. This procedure is known as *pruning*, because the weakest splits in the tree are pruned, leaving only important splits.

Returning to the example of Figure 1-1, we see that the fourth split, applied in Figure 1-1e, creates lower and upper partitions that misclassify 0 and 1 points, respectively. If this split were not applied, the single overall partition would predict "virginica" in line with the most common label, and so the resulting misclassification would be 2 points. This split has therefore only increased the accuracy by a single point, whereas the other splits in the tree give much larger improvements in accuracy, and so pruning the tree will enable us to remove this less significant split, depending on the tradeoff we desire between accuracy and complexity. The original tree is shown in Figure 1-2, and if we prune this tree with the complexity parameter $\alpha = 0.01$, we get the tree shown in Figure 1-3. In the pruned tree, the fourth split has indeed been removed, because the increase in accuracy of 1 point did not outweigh the cost of the additional complexity of another split.

In practice, the value for the complexity parameter would typically be chosen

Figure 1-3: The CART tree for the "Iris" dataset after conducting pruning.



using a hold-out validation set. In this case, we choose the particular value for α that maximizes the accuracy of the tree on this validation set, rather than on the training set, with the goal of choosing the "right-sized tree" that performs best on new unseen data. The standard procedure for tuning the value of α for CART is known as *cost-complexity pruning*, which we will introduce in Section 2.4.

Since we are trying to determine the tree with the best tradeoff between accuracy and interpretability, it is natural to search among the trees of a given complexity for the one with the best accuracy. For instance, using the methodology we develop in Chapter 2, we can verify that the tree in Figure 1-3 is the most accurate tree with three splits: it misclassifies only three points, and there are no other trees with three or fewer splits that have a lower misclassification score. We therefore refer to this tree as the *optimal* tree with three splits. Note that there can be many many such optimal trees with the same number of splits and the minimal misclassification.

A natural extension to decision tree methods like CART is to consider splits that are not parallel to the axes. If we examine the example in Figure 1-1, we see that the CART tree uses two splits to separate the "versicolor" and "virginica" species. However, it seems that these points would also be relatively well-separated by a single diagonal split. Such a partitioning is shown in Figure 1-4, and the corresponding tree that generates these partitions is given in Figure 1-5. This tree with hyperplane splits only misclassifies two points, and is able to better discover the structure in the data, whereas the CART tree is approximating this structure with multiple axis-parallel splits. This results in the tree with hyperplane splits having both higher accuracy

Figure 1-4: An example of partitioning the "Iris" dataset with arbitrary hyperplanes rather than just splits parallel to the axes.

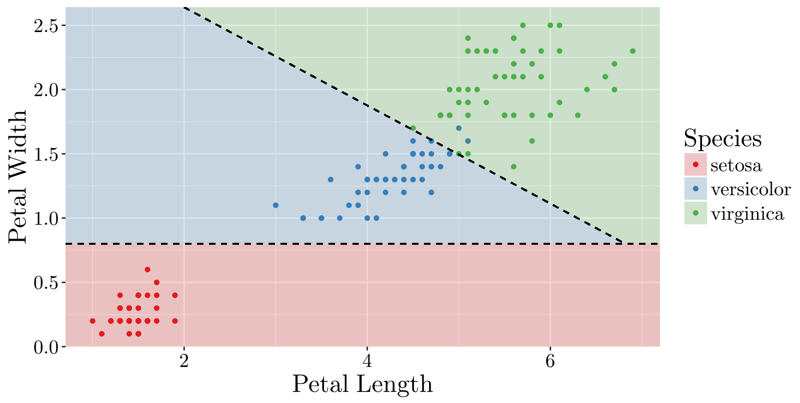
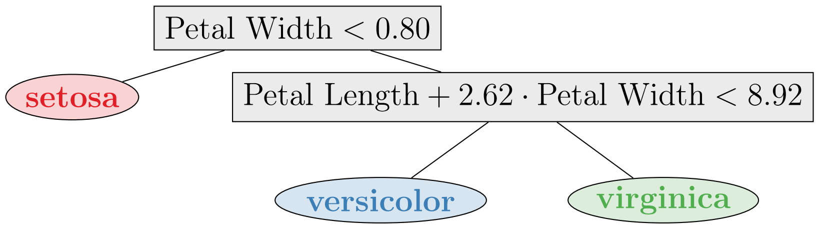


Figure 1-5: A tree for the "Iris" dataset with hyperplane splits.



and one fewer split than the tree with parallel splits. Trees with hyperplane splits are typically more expensive to compute than those with parallel splits, because there are many more possible splits to consider when trying to find the optimal split during the recursion.

1.2 Limitations of CART

The main shortcoming of the top-down approach taken by CART and other popular decision tree methods like C4.5 [104] and ID3 [103], is its fundamentally greedy nature. Each split in the tree is determined in isolation without considering the possible impact of future splits in the tree. This can lead to trees that do not capture well the underlying characteristics of the dataset, potentially leading to weak performance when classifying future points. Figure 1-6 shows an example where a tree grown via top-down induction is very different from the true tree that generated the data.

Another limitation of top-down induction methods is that they typically require

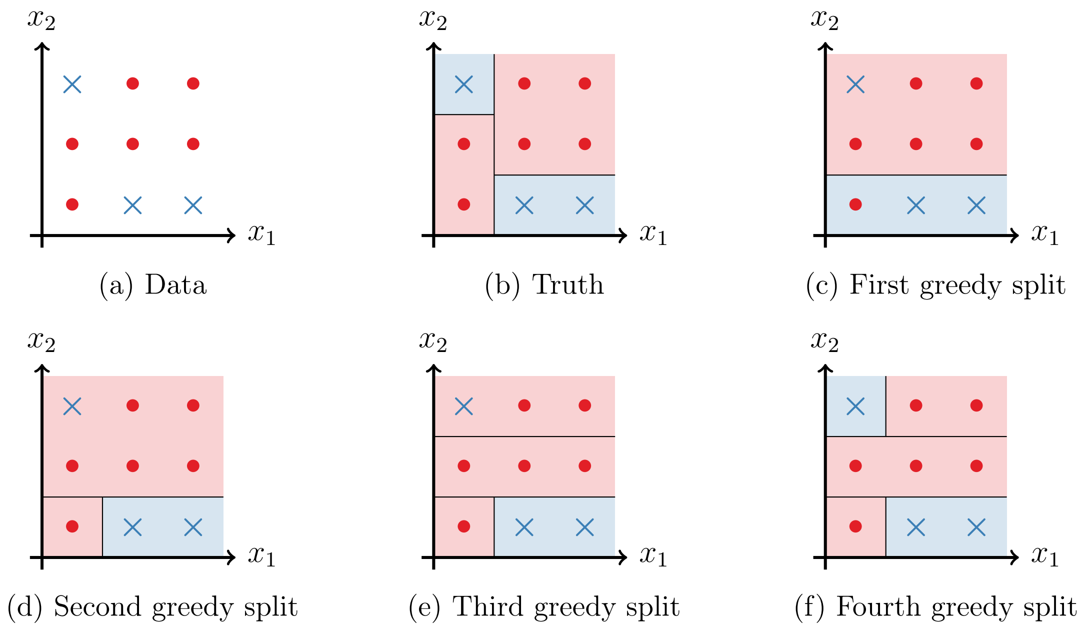


Figure 1-6: An example where greedy tree induction fails to learn the truth in the data. Figure 1-6a shows the data, and the true decision tree that generated the data is given in Figure 1-6b. Figures 1-6c–1-6f show the evolution of a tree created via greedy top-down induction. We see that the greedy induction chooses the wrong split at the very first step, and generates a final tree that has a significantly different structure to the true tree, despite having the same accuracy.

(a) Data (b) Truth (c) First greedy split

(d) Second greedy split (e) Third greedy split (f) Fourth greedy split

pruning to achieve trees that generalize well. Pruning is needed because top-down induction is unable to handle penalties (we call them complexity penalties) on how large the tree becomes while growing it, since powerful splits may be hidden behind weaker splits. This means that the best tree might not be found by the top-down method if the complexity penalty is too high and prevents the first weaker split from being selected.

The usual approach to resolve this is to grow the tree as deep as possible before pruning back up the tree using the complexity penalty. This avoids the problem of weaker splits hiding stronger splits, but it means that the training occurs in two phases, growing via a series of greedy decisions, followed by pruning. Lookahead heuristics such as IDX [94], LSID3 and ID3-k [43] also aim to resolve this problem of strong splits being hidden behind weak splits by finding new splits based on optimizing deeper trees rooted at the current leaf, rather than just optimizing a single split. However, it is unclear whether these methods can lead to trees with better generalization ability and avoid the so-called look-ahead pathology of decision tree learning [89].

In classification problems, top-down induction methods typically optimize an impurity measure when selecting splits, rather than using the misclassification rate of training points. This seems odd when the misclassification rate is the final objective being targeted by the tree, and indeed is also the measure that is used when pruning the tree. Breiman et al. [25, p.97] explain why CART uses impurity measures in place of the more natural objective of misclassification:

> *. . . the [misclassification] criterion does not seem to appropriately reward splits that are more desirable in the context of the continued growth of the tree. . . . This problem is largely caused by the fact that our tree growing structure is based on a one-step optimization procedure.*

Indeed, the misclassification criterion does not always work well within a greedy framework. This was the case in the example from Figure 1-6, where adding a split to Figure 1-6d to get to Figure 1-6e improves the impurity measure, but does not improve the misclassification error, and so we would be unable to proceed past the second greedy split if we were simply using the misclassification criterion. It seems natural to believe that growing the decision tree with respect to the final objective function would lead to better splits, but the use of top-down induction methods and their requirement for pruning prevents the use of this objective.

The natural way to address the limitations of CART is to form the entire decision tree in a single step, allowing each split to be determined with full knowledge of all other splits. This would result in the *optimal* decision tree for the training data. This is not a new idea; Breiman et al. [25, p.42] noted the potential for such a method:

> *Finally, another problem frequently mentioned (by others, not by us) is that the tree procedure is only one-step optimal and not overall optimal . . . If one could search all possible partitions . . . the two results might be quite different . . . At this stage of computer technology, an overall optimal*

tree growing procedure does not appear feasible for any reasonably sized dataset.

The use of top-down induction and pruning in CART was therefore not due to a belief that such a procedure was inherently better, but instead was guided by practical limitations of the time, given the difficulty of finding an optimal tree. Indeed, it is well-known that the problem of constructing optimal binary decision trees is NP-hard [64]. Nevertheless, there have been many efforts previously to develop effective ways of constructing optimal univariate decision trees using a variety of heuristic methods for growing the tree in one step, including linear optimization [7], continuous optimization [8], dynamic programming [37, 99], genetic algorithms [109], and more recently, evolutionary algorithms [55], and optimizing an upper bound on the tree error using stochastic gradient descent [93]. However, none of these methods have been able to produce certifiably optimal trees in practical times. A different approach is taken by the methods T2 [4], T3 [112] and T3C [117], a family of efficient enumeration approaches which create optimal non-binary decision trees of depths up to 3. However, trees produced using these enumeration schemes are not as interpretable as binary decision trees, and do not perform significantly better than current heuristic approaches [112, 117].

1.3 The Remarkable Progress of MIO

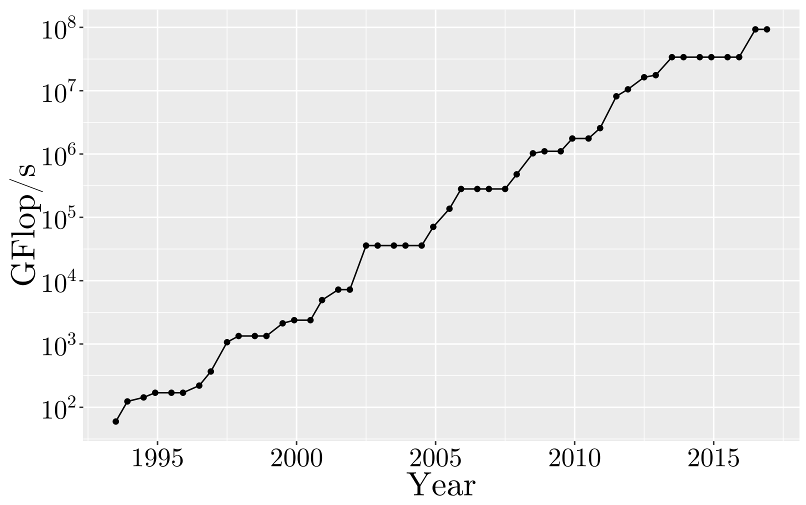
We believe the lack of success in developing an algorithm for optimal decision trees can be attributed to a failure to address correctly the underlying nature of the decision tree problem. Constructing a decision tree involves a series of discrete decisions—whether to split at a node in the tree, which variable to split on—and discrete outcomes—which leaf node a point falls into, whether a point is correctly classified in a classification problem—and as such, the problem of creating an optimal decision tree is best posed as a Mixed-Integer Optimization (MIO) problem.

Continuous optimization methods have been widely used in statistics over the past 40 years, but MIO methods, which have been used to great effect in many other fields, have not had the same impact on statistics. Despite the knowledge that many statistical problems have natural MIO formulations [1], there is the belief within the statistics/machine learning community that MIO problems are intractable even for small to medium instances, which was true in the early 1970s when the first continuous optimization methods for statistics were being developed.

However, the last twenty-five years have seen an incredible increase in the computational power of MIO solvers, and modern MIO solvers such as Gurobi [56] and CPLEX [65] are able to solve linear MIO problems of considerable size. To quantify this increase, Bixby [22] tested a set of MIO problems on the same computer using CPLEX 1.2, released in 1991, through CPLEX 11, released in 2007. The total speedup factor was measured to be more than 29,000 between these versions [22, 92]. Gurobi 1.0, an MIO solver first released in 2009, was measured to have similar performance to CPLEX 11. Version-on-version speed comparisons of successive Gurobi releases have shown a speedup factor of around 43 between Gurobi 7.0, released in 2016, and Gurobi 1.0 [22, 92, 57], so the combined machine-independent speedup factor in MIO solvers between 1991 and 2015 is approximately 1,250,000. This impressive speedup factor is due to incorporating both theoretical and practical advances into MIO solvers. Cutting plane theory, disjunctive programming for branching rules, improved heuristic methods, techniques for preprocessing MIOs, using linear optimization as a black box to be called by MIO solvers, and improved linear optimization methods have all contributed greatly to the speed improvements in MIO solvers [22]. Coupled with the increase in computer hardware during this same period as shown in Figure 1-7, a factor of approximately 1,560,000 [113], the overall speedup factor is approximately 2 trillion! This astonishing increase in MIO solver performance has enabled many recent successes when applying modern MIO methods to a selection of these statistical problems [18, 13, 14, 15].

The belief that MIO approaches to problems in statistics are not practically relevant was formed in the 1970s and 1980s and it was at the time justified. Given the astonishing speedup of MIO solvers and computer hardware in the last twenty-five years, the mindset of MIO as theoretically elegant but practically irrelevant is no

Figure 1-7: Peak supercomputer speed in GFlop/s (log scale) from 1994–2016.



longer supported. In this thesis, we extend this re-examination of statistics under a modern optimization lens by using MIO to formulate and solve the decision tree training problem, and provide empirical evidence of the success of this approach.

1.4 What is Tractable?

The computer science community has developed the notion that problems that are solvable by an algorithm in polynomial time (in the bits to write the input of the instance of a problem), that is they belong to the class P, are *theoretically tractable*. In contrast, the theory of NP-completeness, see Garey and Johnson [51], has been developed in the 1970s to explain that a problem that is NP-hard can not be solved in polynomial time unless $P = NP$. While is still unknown whether $P = NP$, it is widely believed that $P \neq NP$. Thus, the current belief is that theoretical tractability is equivalent to polynomial time solvability. In this way, optimal trees being NP-hard are believed to be theoretically intractable.

It is our belief, however, that a 2 trillion time speedup forces us to re-consider what is practically tractable. To motivate our discussion, let us give two examples. The simplex method developed by George Dantzig [38, 39] for linear optimization problems is not a polynomial time algorithm (for many variants of the method there are instances that require exponential time to find an optimal solution). Yet, to this day it is an extremely practical algorithm that is being used widely in practice for sizes involving millions of variables and constraints. In the same way, the traveling salesman problem is NP-hard, yet problems with one million cities can be routinely solved in minutes. In other words, the predictions of complexity theory regarding tractability have often negative correlation with empirical evidence.

For this reason, we define the notion of *practical tractability (practability)*. A problem is practically tractable (practable) if it can be solved in times and for sizes that are appropriate for the application that motivated the problem. Let us give some examples. What is relevant for an online trading problem is our ability to solve it in milliseconds for sizes of 500–1000 securities. An optimal tree for medical diagnosis needs to be able to be constructed in hours or days for sizes involving hundreds of thousands of patients, but it should be able to run online in seconds to diagnose a new patient. What is important is that polynomial solvability or NP-hardness does not give relevant information for our ability to solve the problem in the real world. Given that the motivation of this thesis is to solve real world problems, we will use the notion of practical tractability alongside theoretical tractability when evaluating our algorithms.

1.5 Motivation, Philosophy and Goal

Decision trees have good but, because of the limitations we discussed, not excellent accuracy for prediction problems. For this reason Leo Breiman [29] introduced random forests, also extremely influential in mathematical sciences, having been cited 26,144 times (as of February 13, 2017). The idea of random forests is to create a large collection of trees each trained on a random collection of features and on a random subset of the training data. The final prediction is based on a majority vote among the trees in the forest for classification, and the mean prediction for regression. The accuracy of random forests improves upon the accuracy of a single tree, but at the expense of loss of interpretability. Jerry Friedman [49] proposed the idea of gradient boosting, which develops a collection of sequentially generated trees where tree T_{k+1} is trained to predict the residuals of the collection of trees $T_1, \ldots, T_k$. Boosted trees have comparable or slightly better accuracy compared to random forests, but also sacrifice interpretability as they use a collection of trees to make a prediction as opposed to a single tree.

Given the success of MIO, our discussion of practical tractability and of the state of the art of decision trees, natural questions arise:

1. Given that decision trees are found by greedy methods, can we find globally optimal trees in a practically tractable way?

2. Can we construct optimal trees in a practically tractable way in which we use hyperplanes splits to partition the space?

3. Do optimal trees with either parallel or hyperplane splits lead to significantly improved accuracy compared to CART?

4. How does the accuracy of optimal trees compare to random forests and boosted trees?

5. Can we construct optimal regression trees in a practically tractable way in which we use either parallel or hyperplane splits and the prediction in each final partition (a leaf in the tree) is an affine function of the features, similar to linear regression?

6. Decision trees have been used only for prediction. Can we develop optimal classification and regression trees for prescription?

7. Most importantly, do optimal trees provide a significant advantage in real world settings?

Motivated by these questions we present in this thesis a theory for constructing in a practically tractable way optimal trees for prediction and prescriptions. In this effort, we are guided by the words of George Dantzig [39], who in the opening sentence of his book "Linear Programming and Extensions" writes

The final test of any theory is its capacity to solve the problems which originated it.

Specifically, in the following chapters we show

1. Optimal or near-optimal trees for classification and regression with either parallel or hyperplane splits can be constructed in a both theoretically and practically tractable way.

2. Optimal trees with parallel splits improve the prediction accuracy of CART, while still maintaining interpretability.

3. Optimal trees with hyperplane splits significantly improve the prediction accuracy of CART. We find that the accuracy of optimal trees with hyperplane splits is comparable to random forests and boosted trees. We feel this is materially relevant, since with one interpretable tree, we match or improve upon methods with state-of-the-art accuracy that require a large collection of trees, which are not interpretable.

4. We construct optimal prescription trees that advances machine learning in the direction of optimal decisions rather than predictions.

5. We provide evidence from real world applications that our methods provide significant advantages in accuracy or interpretability or both.

Our overall philosophy is to emphasize practical tractability and impact on real-world applications.

1.6 Structure of the Thesis

A chapter by chapter description of the thesis is as follows.

Chapter 2: Introduces optimal classification trees using parallel splits, provides solutions derived both using MIO and local improvement methods and presents results on accuracy in both synthetic and real world data sets.

Chapter 3: Discusses optimal classification trees using hyperplane splits and emphasizes how the method compares with random forests and boosted trees using both real and synthetic data sets.

Chapter 4: Contains optimal regression trees with constant predictions, where the prediction in each leaf of the tree is the average of all the values of the dependent variable among all data points that are included in the leaf of the tree, and compares how this approach improves upon the CART methodology using real and synthetic data.

Chapter 5: Deals with optimal regression trees with linear predictions, where the prediction in each leaf of the tree comes from a linear regression involving all the points that are included in the leaf, and presents evidence that they lead to significantly improved accuracy.

Chapter 6: Includes a discussion of optimal prescription trees that are generalizations of the prediction trees we presented in earlier chapters and aim to construct trees that lead to optimal decisions, and provides several examples that suggest that prescription trees provide an edge in taking optimal decisions.

Chapter 7: Provides a real world example of using optimal classification trees to diagnose head trauma that illustrate the edge these methods have over alternatives.

Chapter 8: Summarizes the key messages of the thesis and includes some closing thoughts.

Notation

Throughout the thesis, boldfaced lowercase letters $(\mathbf{x}, \mathbf{y}, \ldots)$ denote vectors, boldfaced capital letters $(\mathbf{X}, \mathbf{Y}, \ldots)$ denote matrices, and ordinary lowercase letters (x, y) denote scalars. Calligraphic type $(\mathcal{P}, \mathcal{S}, \ldots)$ denotes sets. The notation $[n]$ is used as shorthand for the set $\{1, \ldots n\}$.

Chapter 2

Optimal Classification Trees with Parallel Splits

In this chapter, we demonstrate that formulating the decision tree problem using MIO leads to a tractable approach and delivers practical solutions that significantly outperform classical approaches.

We summarize the main results of this chapter below:

1. We present a new, novel formulation of the classical decision tree problem as an MIO problem that motivates our new classification method, *Optimal Classification Trees* (OCT).

2. Using a range of tests with synthetic data comparing OCT against CART, we demonstrate that solving the decision tree problem to optimality yields trees that better reflect the ground truth in the data, refuting the belief that such optimal methods will simply overfit to the training set and not generalize well.

3. We demonstrate that our OCT method outperforms classical decision tree methods in practical applications. We comprehensively benchmark OCT against the state-of-the-art CART on a sample of 60 datasets from the UCI Machine Learning Repository. We show that OCT yields higher out-of-sample accuracy than CART, with average improvements of 1–2% over CART across all datasets,

depending on the depth of tree used, and that this difference is statistically significant at all depths.

4. To provide guidance to machine learning practitioners, we present a simple decision rule that predicts when OCT will offer the largest accuracy improvements over CART. If the number of points is small or the number of features is large, this rule is satisfied indicating the problem has high difficulty. Across our sample of datasets where this rule is satisfied, OCT improves upon CART by an average of 2.59%, otherwise the mean improvement is 0.26%.

We note that there have been efforts in the past to apply MIO methods to classification problems. CRIO (Classification and Regression via Integer Optimization), proposed by Bertsimas and Shioda [16], uses MIO to partition and classify the data points. CRIO was not able to solve the classification problems to provable optimality for moderately-sized classification problems, and the practical improvement over CART was not significant. In contrast, in this thesis we present a very different MIO approach based around solving the same problem that CART seeks to solve, and this approach provides material improvement over CART for a variety of datasets.

2.1 Review of Classification Tree Methods

We are given the training data $(\mathbf{X}, \mathbf{y})$, containing n observations $(\mathbf{x}_i, y_i)$, $i \in [n]$, each with p features $\mathbf{x}_i \in \mathbb{R}^p$ and a class label $y_i \in [K]$ indicating which of K possible labels is assigned to this point. We assume without loss of generality that the values for each dimension across the training data are normalized to the 0–1 interval, meaning each $\mathbf{x}_i \in [0, 1]^p$.

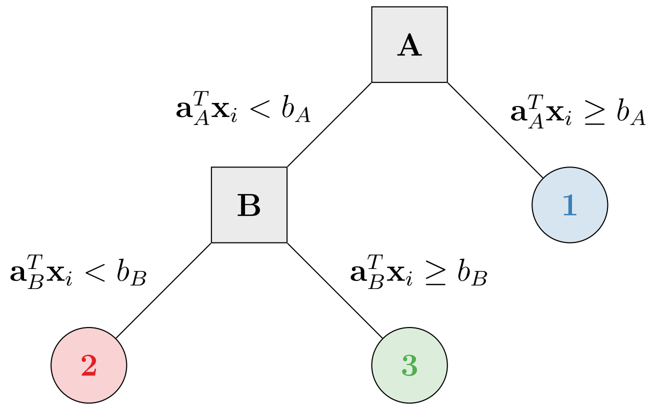
Decision tree methods seek to recursively partition $[0, 1]^p$ to yield a number of hierarchical, disjoint regions that represent a classification tree. An example of a decision tree is shown in Figure 2-1. The final tree is comprised of branch nodes and leaf nodes:

- Branch nodes apply a split with parameters $\mathbf{a}$ and b. For a given point i, if

$\mathbf{a}^T\mathbf{x}_i < b$ the point will follow the left branch from the node, otherwise it takes the right branch. Most classical methods, including CART, produce *univariate* or *axis-aligned* decision trees which restrict the split to a single dimension, i.e., a single component of $\mathbf{a}$ will be 1 and all others will be 0.

- Leaf nodes are assigned a class that will determine the prediction for all data points that fall into the leaf node. The assigned class is usually taken to be the most-common class among points contained in the leaf node.

Figure 2-1: An example of a decision tree with two branch nodes, A and B, and three leaf nodes, 1, 2 and 3.


A
$\mathbf{a}_A^T\mathbf{x}_i < b_A$
$\mathbf{a}_A^T\mathbf{x}_i \geq b_A$
B
1
$\mathbf{a}_B^T\mathbf{x}_i < b_B$
$\mathbf{a}_B^T\mathbf{x}_i \geq b_B$
2
3

Classical decision tree methods like CART, ID3, and C4.5 take a top-down approach to building the tree. At each step of the partitioning process, they seek to find a split that will partition the current region in such a way to maximize a so-called splitting criterion. This criterion is often based on the label impurity of the data points contained in the resulting regions instead of minimizing the resulting misclassification error, which as discussed earlier is a byproduct of using a top-down induction process to grow the tree. The algorithm proceeds to recursively partition the two new regions that are created by the hyperplane split. The partitioning terminates once any one of a number of stopping criteria are met. The criteria for CART are as follows:

- It is not possible to create a split where each side of the partition has at least a certain number of nodes, $N_{\min}$;

- All points in the candidate node share the same class.

Once the splitting process is complete, a class label $c \in [K]$ is assigned to each region. This class will be used to predict the class of any points contained inside the region. As mentioned earlier, this assigned class will typically be the most common class among the points in the region.

The final step in the process is pruning the tree in an attempt to avoid overfitting. The pruning process works upwards through the partition nodes from the bottom of the tree. The decision of whether to prune a node is controlled by the so-called *complexity parameter*, denoted by α, which balances the additional complexity of adding the split at the node against the increase in predictive accuracy that it offers. A higher complexity parameter leads to more and more nodes being pruned off, resulting in smaller trees.

As discussed previously, this two-stage growing and pruning procedure for creating the tree is required when using a top-down induction approach, as otherwise the penalties on tree complexity may prevent the method from selecting a weaker split that then allows a selection of a stronger split in the next stage. In this sense, the strong split is hidden behind the weak split, and this strong split may be passed over if the growing-then-pruning approach is not used. An example of this phenomenon is shown in Figure 2-2

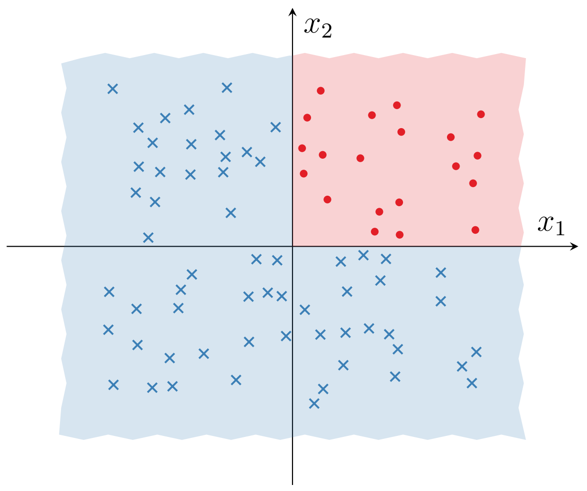
Using the details of the CART procedure, we can state the problem that CART attempts to solve as a formal optimization problem. There are two parameters in this problem. The tradeoff between accuracy and complexity of the tree is controlled by the complexity parameter α, and the minimum number of points we require in any leaf node is given by $N_{\min}$. Given these parameters and the training data $(\mathbf{x}_i, y_i), i \in [n]$, we seek a tree $\mathbb{T}$ that solves the problem:

$$\begin{aligned} \min \quad & R(\mathbb{T}) + \alpha|\mathbb{T}| \\ \text{s.t.} \quad & N(\ell) \geq N_{\min} \qquad \forall \ell \in \text{leaves}(\mathbb{T}) \end{aligned} \tag{2.1}$$

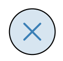
where $R(\mathbb{T})$ is the misclassification error of the tree $\mathbb{T}$ on the training data, $|\mathbb{T}|$ is the number of branch nodes in tree $\mathbb{T}$, and $N(\ell)$ is the number of training points

Figure 2-2: An example of a strong split being hidden behind a weak split, motivating the grow-then-prune approach to tree construction. The data in Figure 2-2a is generated according to a simple `AND` boolean rule: points in the positive quadrant have label ●, otherwise they are labeled ×. Figures 2-2b–2-2d show three possible classification trees for this data, in increasing order of tree complexity, showing the growth of the trees according to a top-down induction. The complexity 0 tree has no splits and simply predicts the majority class, with an error of 25%. Adding the single best split gives the complexity 1 tree, but does not improve the error rate. Adding the best second split gives the complexity 2 tree, which encodes the `AND` relationship exactly and reduces the error to 0%. The complexity 1 tree does not improve upon the error of the complexity 0 tree, and so this split will not overcome any penalty from the complexity parameter and thus does not lower the overall objective. This causes the induction process to terminate with the complexity 0 tree as the best solution, despite the accuracy of the complexity 2 tree. The second split is only valuable after the first split has been applied, and therefore is hidden behind the first split and cannot be reached by the greedy induction process. Growing and then pruning the tree in a two-stage process mitigates this limitation, as the tree is grown to full depth before considering whether the splits actually decrease the overall objective during pruning.

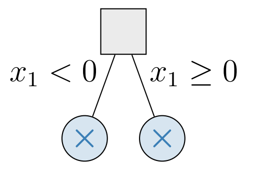
(a) Data



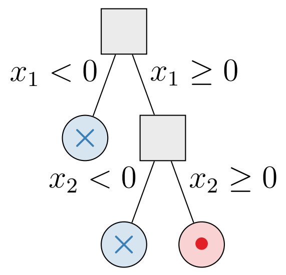
(b) Comp. 0: 25% error



(c) Comp. 1: 25% error



(d) Comp. 2: 0% error



contained in leaf node ℓ. We refer to this problem as the *optimal tree problem.*

Notice that if we can solve this problem in a single step we obviate the need to use an impurity measure when growing the tree, and also remove the need to prune the tree after creation, as we have already accounted for the complexity penalty while growing the tree.

We briefly note that our choice to use CART to define the optimal tree problem was arbitrary, and one could similarly define this problem based on another method like C4.5; we simply use this problem to demonstrate the advantages of taking a problem that is traditionally solved by a heuristic and instead solving it to optimality. Additionally, we note that the experiments of [90] found that CART and C4.5 did not differ significantly in any measure of tree quality, including out-of-sample accuracy, and so we do not believe our choice of CART over C4.5 to be an important one.

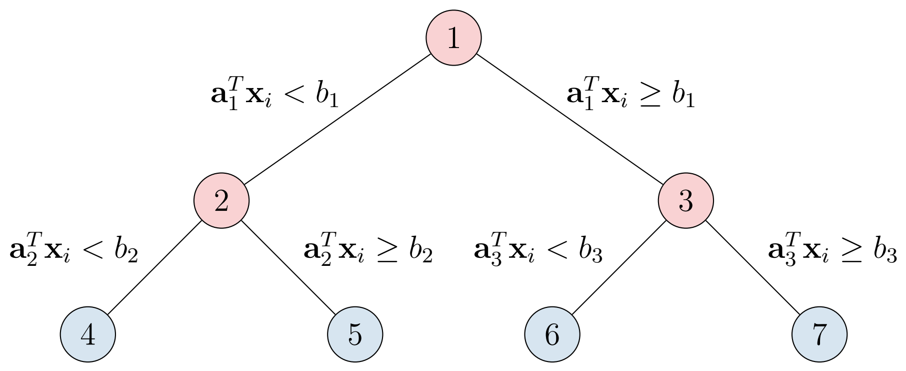
2.2 A Mixed-Integer Optimization Approach

As mentioned previously, the top-down, greedy nature of state-of-the-art decision tree creation algorithms can lead to solutions that are only locally optimal. In this section, we first argue that the natural way to pose the task of creating the globally optimal decision tree is as an MIO problem, and then proceed to develop such a formulation.

To see that the most natural representation for formulating the optimal decision tree problem (2.1) is using MIO, we note that at every step in tree creation, we are required to make a number of discrete decisions:

- At every new node, we must choose to either branch or stop.
- After choosing to stop branching at a node, we must choose a label to assign to this new leaf node.
- After choosing to branch, we must choose which of the variables to branch on.
- When classifying the training points according to the tree under construction, we must choose to which leaf node a point will be assigned such that the structure of the tree is respected.

Figure 2-3: The maximal tree for depth $D = 2$. Branch nodes are red and leaf nodes are blue.


1
$\mathbf{a}_1^T\mathbf{x}_i < b_1$
$\mathbf{a}_1^T\mathbf{x}_i \geq b_1$
2
3
$\mathbf{a}_2^T\mathbf{x}_i < b_2$
$\mathbf{a}_2^T\mathbf{x}_i \geq b_2$
$\mathbf{a}_3^T\mathbf{x}_i < b_3$
$\mathbf{a}_3^T\mathbf{x}_i \geq b_3$
4
5
6
7

Formulating this problem using MIO allows us to model all of these discrete decisions in a single problem, as opposed to recursive, top-down methods that must consider these decision events in isolation. Modeling the construction process in this way allows us to consider the full impact of the decisions being made at the top of the tree, rather than simply making a series of locally optimal decisions, also avoiding the need for pruning and impurity measures.

We will next formulate the optimal tree creation problem (2.1) as an MIO problem. Consider the problem of trying to construct an optimal decision tree with a maximum depth of D. Given this depth, we can construct the *maximal tree* of this depth with $T = 2^{(D+1)} - 1$ nodes, which we index by $t \in [T]$. Figure 2-3 shows the maximal tree of depth 2.

We use the notation $p(t)$ to refer to the parent node of node t, and $\mathcal{A}(t)$ to denote the set of ancestors of node t. We also define $\mathcal{L}(t)$ as the set of ancestors of t whose left branch has been followed on the path from the root node to t, and similarly $\mathcal{R}(t)$ is the set of right-branch ancestors, such that $\mathcal{A}(t) = \mathcal{L}(t) \cup \mathcal{R}(t)$. For example, in the tree in Figure 2-3, $A_L(5) = \{1\}$, $A_R(5) = \{2\}$, and $A(5) = \{1, 2\}$.

We divide the nodes in the tree into two sets:

Branch nodes: Nodes $t \in \mathcal{T}_B = \{1, \ldots, \lfloor T/2 \rfloor\}$ apply a split of the form $\mathbf{a}^\mathsf{T}\mathbf{x} < b$. Points that satisfy this split follow the left branch in the tree, and those that do not follow the right branch.

Leaf nodes: Nodes $t \in \mathcal{T}_L = \{\lfloor T/2 \rfloor + 1, \dots, T\}$ make a class prediction for each point that falls into the leaf node.

We track the split applied at node $t \in \mathcal{T}_B$ with variables $\mathbf{a}_t \in \mathbb{R}^p$ and $b_t \in \mathbb{R}$. In this chapter, we restrict our model to univariate decision trees (like CART), and so the hyperplane split at each node should only involve a single variable. This is enforced by setting the elements of $\mathbf{a}_t$ to be binary variables that sum to 1. We want to allow the option of not splitting at a branch node. We use the binary variables d_t to track which branch nodes apply splits, where $d_t = 1$ if node t applies a split, and $d_t = 0$ otherwise. If a branch node does not apply a split, then we model this by setting $\mathbf{a}_t = \mathbf{0}$ and $b_t = 0$. This has the effect of forcing all points to follow the right split at this node, since the condition for the left split is $0 < 0$ which is never satisfied. This allows us to stop growing the tree early without introducing new variables to account for points ending up at the branch node—instead we send them all the same direction down the tree to end up in the same leaf node. We enforce this with the following constraints:

$$\sum_{j=1}^{p} a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B, \tag{2.2}$$

$$0 \le b_t \le d_t, \quad \forall t \in \mathcal{T}_B, \tag{2.3}$$

$$a_{jt} \in \{0,1\}, \quad \forall j \in [p],\ t \in \mathcal{T}_B, \tag{2.4}$$

where the second inequality is valid for b_t, since we have assumed that each $\mathbf{x}_i \in [0,1]^p$, and we know that $\mathbf{a}_t$ has one element that is 1 if and only if $d_t = 1$, with the remainder being 0. Therefore, it is always true that $0 \le \mathbf{a}_t^\mathsf{T} \mathbf{x}_i \le d_t$ for any i and t, and we need only consider values for b_t in this same range.

Next, we will enforce the hierarchical structure of the tree. We restrict a branch node from applying a split if its parent does not also apply a split.

$$d_t \le d_{p(t)}, \quad \forall t \in \mathcal{T}_B \setminus \{1\}, \tag{2.5}$$

where no such constraint is required for the root node.

We have constructed the variables that allow us to model the tree structure using MIO; we now need to track the allocation of points to leaves and the associated errors that are induced by this structure.

We introduce binary variables z_{it} to track the points assigned to each leaf node, where $z_{it} = 1$ if point i is in node t, and $z_{it} = 0$ otherwise. We also introduce binary variables l_t, where $l_t = 1$ if leaf t contains any points, and $l_t = 0$ otherwise. We use these binary variables together to enforce a minimum number of points at each leaf, given by $N_{\min}$:

$$z_{it} \leq l_t, \qquad \forall t \in \mathcal{T}_B, \tag{2.6}$$

$$\sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, \quad \forall t \in \mathcal{T}_B. \tag{2.7}$$

We also force each point to be assigned to exactly one leaf:

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad \forall i \in [n]. \tag{2.8}$$

Finally, we apply constraints enforcing the splits that are required by the structure of the tree when assigning points to leaves:

$$\mathbf{a}_m^\mathsf{T} \mathbf{x}_i < b_m + M_1(1 - z_{it}), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t), \tag{2.9}$$

$$\mathbf{a}_m^\mathsf{T} \mathbf{x}_i \geq b_m - M_2(1 - z_{it}), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t), \tag{2.10}$$

where M_1 and M_2 are sufficiently large constants such that the constraints are always satisfied when $z_{it} = 0$. We will discuss choosing values for these constants shortly, but first note that the constraints (2.9) use a strict inequality. This is not supported by MIO solvers, and so it must be converted into a form that does not use a strict inequality. To do this we can add a small constant ϵ to the left-hand-side of (2.9) and change the inequality to be non-strict:

$$\mathbf{a}_m^T\mathbf{x}_i + \epsilon \leq b_m + M_1\left(1 - z_{it}\right), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t). \tag{2.11}$$

However, if ϵ is too small, this could cause numerical instabilities in the MIO solver, so we seek to make ϵ as big as possible without affecting the feasibility of any valid solution to the problem. We can achieve this by specifying a different ϵ_j for each feature j. The largest valid value is the smallest non-zero distance between adjacent values of this feature. To find this, we sort the values of the jth feature and take

$$\epsilon_j = \min\left\{ x_j^{(i+1)} - x_j^{(i)} \;\middle|\; x_j^{(i+1)} \neq x_j^{(i)}, i \in [n-1] \right\},$$

where $x_j^{(i)}$ is the ith largest value in the jth feature. We can then use these values for ϵ in the constraint, where the value of ϵ_j that is used is selected according to the feature we are using for this split:

$$\mathbf{a}_m^T(\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + M_1\left(1 - z_{it}\right), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t). \tag{2.12}$$

We must also specify values for the big-M constants M_1 and M_2. As mentioned previously, we know that both $\mathbf{a}_t^T\mathbf{x}_i \in [0,1]$ and $b_t \in [0,1]$, and so the largest possible value of $\mathbf{a}_t^T(\mathbf{x}_i + \epsilon) - b_t$ is $1 + \epsilon_{\max}$, where $\epsilon_{\max} = \max_j\{\epsilon_j\}$. We can therefore set $M_1 = 1 + \epsilon_{\max}$. Similarly, we have the largest possible value of $b_t - \mathbf{a}_t^T x_i$ is 1, and so we can set $M_2 = 1$. This gives the following final constraints that will enforce the splits in the tree:

$$\mathbf{a}_m^T(\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})\left(1 - z_{it}\right), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t), \tag{2.13}$$

$$\mathbf{a}_m^\mathsf{T}\mathbf{x}_i \geq b_m - (1 - z_{it}), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t). \tag{2.14}$$

The objective is to minimize the misclassification error, so an incorrect label prediction has cost 1, and a correct label prediction has cost 0. We set N_{kt} to be the number of points of label k in node t, and N_t to be the total number of points in node

t:

$$N_{kt} = \sum_{i:\ y_i = k} z_{it}, \quad \forall t \in \mathcal{T}_L,\ k \in [K], \tag{2.15}$$

$$N_t = \sum_{i=1}^{n} z_{it}, \qquad \forall t \in \mathcal{T}_L. \tag{2.16}$$

We need to assign a label to each leaf node t in the tree, which we denote with $c_t \in \{1, \ldots, K\}$. It is clear that the optimal label to predict is the most common of the labels among all points assigned to the node:

$$c_t = \underset{k \in [K]}{\arg\max} \{N_{kt}\}. \tag{2.17}$$

We will use binary variables c_{kt} to track the prediction of each node, where $c_{kt} = \mathbb{1}\{c_t = k\}$. We must make a single class prediction at each leaf node that contains points:

$$\sum_{k=1}^{K} c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L. \tag{2.18}$$

Since we know how to make the optimal prediction at each leaf t using (2.17), the optimal misclassification loss in each node, denoted L_t is going to be equal to the number of points in the node less the number of points of the most common label:

$$L_t = N_t - \max_{k \in [K]} \{N_{kt}\} = \min_{k \in [K]} \{N_t - N_{kt}\}, \tag{2.19}$$

which can be linearized to give

$$L_t \geq N_t - N_{kt} - M(1 - c_{kt}), \quad \forall t \in \mathcal{T}_L,\ k \in [K], \tag{2.20}$$

$$L_t \leq N_t - N_{kt} + M c_{kt}, \qquad \forall t \in \mathcal{T}_L,\ k \in [K], \tag{2.21}$$

$$L_t \geq 0, \qquad \forall t \in \mathcal{T}_L, \tag{2.22}$$

where again M is a sufficiently large constant that makes the constraint inactive depending on the value of c_{kt}. Here, we can take $M = n$ as a valid value.

The total misclassification cost is therefore $\sum_{t\in\mathcal{T}_L} L_t$, and the complexity C of the tree is the number of splits included in the tree, given by $C = \sum_{t\in\mathcal{T}_B} d_t$. Following CART, we normalize the misclassification against the baseline accuracy, $\hat{L}$, obtained by simply predicting the most popular class for the entire dataset. This makes the effect of α independent of the dataset size. This means the objective from problem (2.1) can be written:

$$\min \quad \frac{1}{\hat{L}} \sum_{t\in\mathcal{T}_L} L_t + \alpha \cdot C. \tag{2.23}$$

Putting all of this together gives the following MIO formulation for problem (2.1), which we call the *Optimal Classification Trees* (OCT) model:

$$\begin{aligned}
\min \quad & \frac{1}{\hat{L}} \sum_{t\in\mathcal{T}_L} L_t + \alpha \cdot C & & \qquad (2.24)\\
\text{s.t.} \quad & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), & & \forall t \in \mathcal{T}_L,\ k \in [K],\\
& L_t \leq N_t - N_{kt} + nc_{kt}, & & \forall t \in \mathcal{T}_L,\ k \in [K],\\
& L_t \geq 0, & & \forall t \in \mathcal{T}_L,\\
& N_{kt} = \sum_{i:\ y_i = k} z_{it}, & & \forall t \in \mathcal{T}_L,\ k \in [K],\\
& N_t = \sum_{i=1}^{n} z_{it}, & & \forall t \in \mathcal{T}_L,\\
& \sum_{k=1}^{K} c_{kt} = l_t, & & \forall t \in \mathcal{T}_L,\\
& C = \sum_{t\in\mathcal{T}_B} d_t, & &\\
& \mathbf{a}_m^\mathsf{T}\mathbf{x}_i \geq b_m - (1 - z_{it}), & & \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t),\\
& \mathbf{a}_m^\mathsf{T}(\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), & & \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t),\\
& \sum_{t\in\mathcal{T}_L} z_{it} = 1, & & \forall i \in [n],\\
& z_{it} \leq l_t, & & \forall t \in \mathcal{T}_L,\\
& \sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, & & \forall t \in \mathcal{T}_L,
\end{aligned}$$

$$\begin{aligned}
&\sum_{j=1}^{p} a_{jt} = d_t, && \forall t \in \mathcal{T}_B,\\
&0 \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B,\\
&d_t \leq d_{p(t)}, && \forall t \in \mathcal{T}_B \setminus \{1\},\\
&z_{it}, l_t, c_{kt} \in \{0,1\}, && \forall i \in [n],\ k \in [K],\ t \in \mathcal{T}_L,\\
&a_{jt}, d_t \in \{0,1\}, && \forall j \in [p],\ t \in \mathcal{T}_B.
\end{aligned}$$

This model as presented is in a form that can be directly solved by any MIO solver. The difficulty of the model is primarily the number of binary variables z_{it}, which is $n \cdot 2^D$. Empirically we observe that we can find high-quality solutions in minutes for depths up to 4 for datasets with thousands of points. Beyond this depth or dataset size, the rate of finding solutions is slower, and more time is required.

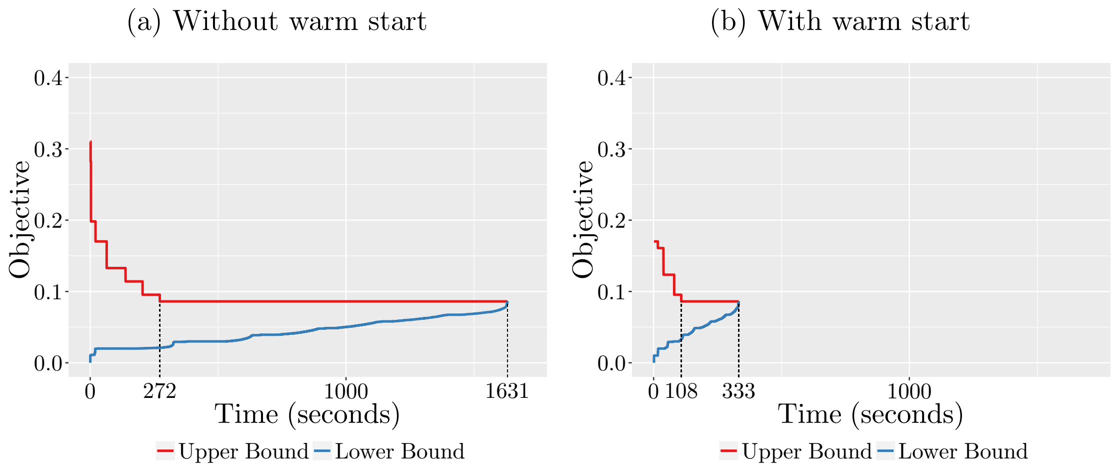
There are three hyper-parameters that need to be specified: the maximum depth D, the minimum leaf size $N_{\min}$, and the complexity parameter α.

Improving MIO Performance using Warm Starts

MIO solvers benefit greatly when supplied an integer-feasible solution as a *warm start* for the solution process. Injecting a strong warm start solution before starting the solver greatly increases the speed with which the solver is able to generate strong feasible solutions [17]. The warm start provides a strong initial upper bound on the optimal solution that allows more of the search tree to be pruned, and it also provides a starting point for local search heuristics. The benefit realized increases with the quality of the warm start, so it is desirable to be able to quickly and heuristically find a strong integer-feasible solution before solving.

We already have access to good heuristic methods for the MIO problem (2.24); we can use CART to generate these warm start solutions. Given a solution from CART, it is simple to construct a corresponding feasible solution to the MIO problem (2.24) using the splits from CART to infer the values of the remaining variables in the MIO problem.

Figure 2-4: Comparison of upper and lower bound evolution while solving MIO problem (2.24) with and without warm starts for a tree of depth $D = 2$ for the Wine dataset with $n = 178$ and $p = 13$.



By design, the parameters N_{min} and α have the same meaning in CART and in problem (2.24), so we can run CART with these parameters to generate a good solution. We must then prune the splits on the CART solution until it is below the maximum depth D for the MIO problem to ensure it is feasible.

We can also use MIO solutions that we have previously generated as warm starts. In particular, if we have a solution generated for a depth D, this solution is a valid warm start for depth $D+1$. This is important because the problem difficulty increases with the depth, so for larger depths it may be advantageous to run the MIO problem with a smaller depth to generate a strong warm start.

To demonstrate the effectiveness of warm starts, Figure 2-4 shows an example of the typical evolution of upper and lower bounds as we solve the MIO problem (2.24) for the optimal tree of depth 2 on the "Wine" dataset, which has $n = 178$ and $p = 13$. We see when no warm start is supplied, the upper bound decreases gradually until the eventual optimal solution is found after 270 seconds. An additional $1,360$ seconds are required to prove the optimality of this solution. When we inject a heuristic solution (in this case, the CART solution) as a warm start, the same optimal solution is found after just 105 seconds, and it takes an additional 230 seconds to prove optimality. The total time required to find and prove optimality in this example decreases by a factor of 5 when the warm start is added, and the time taken to find the optimal solution decreases by a factor of around 2.5, showing that using high-quality warm start solutions can have a significant effect on the MIO solution times. Additionally, we see that the optimal tree solution has an objective with roughly half the error of the CART warm start, showing that the CART solutions can indeed be far from optimality in-sample. Finally, we observe that the majority of the time is spent in proving that the solution is optimal. A proof of optimality is good to have for comparison to other methods, but is not necessarily required in practice when evaluating a classifier (note that other methods make no claims as to their optimality). It is therefore a reasonable option to terminate the solve early once the best solution has remained unchanged for some time, because this typically indicates the solution is indeed optimal or close to it, and proving optimality may take far more time.

2.3 A Local Search Approach

In this section, we investigate and discuss limitations of the MIO-based approach to solving the Optimal Classification Trees problem and use this insight to develop a local-search heuristic for solving the problem that gives better solutions than the MIO approach in a fraction of the time.

Limitations of the MIO approach

The MIO formulation for the Optimal Classification Trees problems as presented in Section 2.2 can be implemented and passed to any mixed-integer optimization solver. However, in our experience the problem is unlikely to solve very fast, and empirically the solutions obtained after letting the solve run for long periods of time can be far from optimality and offer only small improvements over heuristic solutions like those from CART.

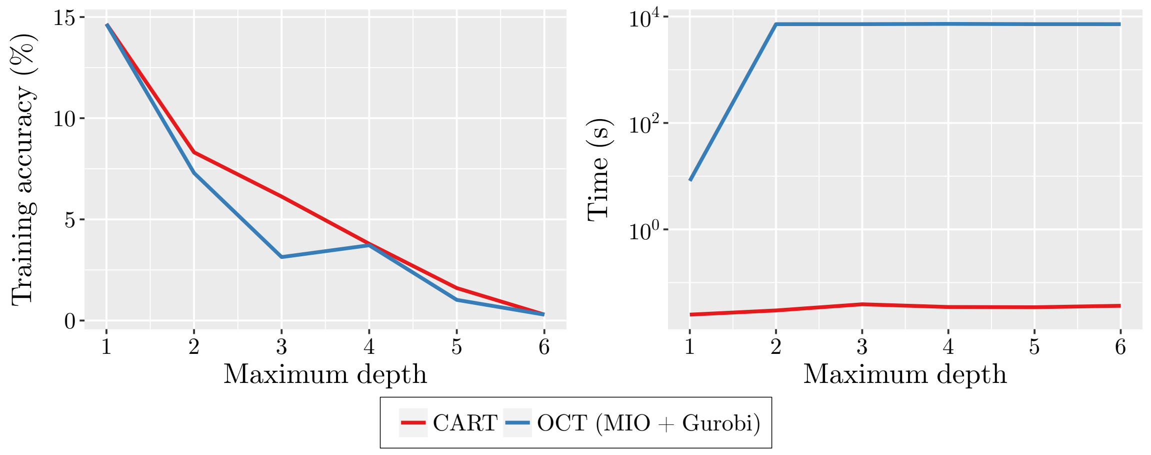
We illustrate this observation with an example using the "Banknote Authentication" dataset from the UCI Machine Learning Repository [77]. This dataset has

$n = 1372$ points, $p = 4$ dimensions, and $K = 2$ classes, so is roughly of low-to-moderate size and difficulty. We compared the performance of CART and OCT on this dataset, measuring the final training error to understand how well each method can solve Problem (2.1). We solved the OCT MIO problem using the Gurobi solver [56], with the CART solution as a warm start for the solution process, and imposed a two hour time limit for the solver to find improved solutions. Figure 2-5 shows the training accuracy of each method on this dataset as a function of the depth of the tree, as well as the corresponding running time. Recall that the size of the MIO problem increases with the maximum depth of the tree we are learning.

We can see that at depth 1, the solutions were identical, and the MIO approach was able to prove optimality after around 10 seconds. At all other depths, no proof of optimality was obtained, and in fact the best MIO lower bound on the training error did not increase from zero after the two hour time limit, and so we have no indication of how close to optimality these solutions might be. OCT was able to improve upon the CART solution in three of the remaining depths, with the largest improvement of about 3% at depth 3. There were no improvements at depths 4 and 6, and at depth 6, Gurobi did not even progress past the root node and begin the branch and bound process within the two hour time limit, due to the sheer size of the problem. In contrast, the CART solutions were found in under one second, so the MIO approach took 5–6 orders of magnitude longer to find solutions that in some cases offered small-to-moderate improvements, and importantly, offered no guarantee that better performance was impossible. This motivates the question of whether better performance is indeed possible, and also motivates a desire to find solutions that improve upon CART in times that are more competitive with CART.

The key reason the MIO problem is slow to solve is the sheer number of variables and constraints that arise in the formulation. In particular, the number of binary variables grows very fast with the number of training points and the depth of the tree; specifically there are $n \cdot 2^D$ binary variables z_{it} to track the potential allocation of each point i to each leaf t. For example, there are $1372 \cdot 2^6 = 87,808$ such binary variables for the largest case in the"Banknote Authentication" example above, which

Figure 2-5: Training accuracy (%) and running time for each method on "Banknote Authentication" dataset.



already proves hard to solve in hours. A typical real-world example might have n in the tens or hundreds of thousands and depths larger than 10, giving over 100 million binary variables! Based on experiments with synthetically-generated data, we find that doubling the number of these z_{it} variables (by increasing the depth by 1 or by doubling n) increases the MIO solve time by roughly an order of magnitude. This makes it unlikely that an MIO-based solution approach would be able to scale to the problem sizes that we would want to solve.

The natural way to resolve this scaling issue is to try to reduce the problem size. One observation we can make is that the "core" decision variables that define a tree are the split variables $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_{\lfloor T/2 \rfloor}\}$ and $\mathbf{b} = \{b_1, \ldots, b_{\lfloor T/2 \rfloor}\}$. Once we are given these split variables, we can solve for the values of the remaining variables and evaluate the error in closed-form, simply by dividing the points according to the splits and then assigning the labels to each leaf using majority-rule. We can frame this problem as follows:

$$\min_{\mathbf{A},\mathbf{b}} \quad \texttt{error}(\mathbf{A}, \mathbf{b}, \mathbf{X}, \mathbf{y}), \tag{2.25}$$

where $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $\mathbf{y} = \{y_1, \ldots, y_n\}$ are the training points and their labels, and `error` is a function that evaluates the objective cost of the splits $\mathbf{A}$ and $\mathbf{b}$ on the training data $\mathbf{X}$ and $\mathbf{y}$.

Immediately, we can see that the number of variables in problem (2.25) is greatly reduced compared to problem (2.24); we only have to optimize over $\mathbf{A} \in \{0,1\}^{n \times \lfloor T/2 \rfloor}$ and $\mathbf{b} \in \mathbb{R}^{\lfloor T/2 \rfloor}$. Most importantly, the number of variables no longer depends on the number of training points n. The complication comes from the non-linear function `error` being embedded in the objective, which means we cannot solve it using standard MIO optimization methods. However, we would expect to solve this problem very fast if we had a way of optimizing this function over the reduced decision space.

Optimization via local search

We will now present the algorithm we have developed for solving Problem (2.25). This local search procedure iteratively improves and refines the current solution until a local minimum is reached. This procedure is repeated from a number of different starting points with the goal of finding many different local minima in the expectation that the best of these is the global minimum, or very close to it.

We begin with some notation and a description of functions used in the presentation of the algorithm:

- $\mathbb{T}_t$ denotes the subtree rooted at the tth node of a tree $\mathbb{T}$;
- $\mathbf{X}_{\mathcal{A}}$ and $\mathbf{y}_{\mathcal{A}}$ denote the subsets of the training data $\mathbf{X}$ and $\mathbf{y}$ corresponding to any index set $\mathcal{A}$;
- shuffle($\mathcal{A}$) randomizes the order of an index set $\mathcal{A}$;
- nodes($\mathbb{T}$) returns a set containing the indices of all nodes in tree $\mathbb{T}$;
- children($\mathbb{T}$) returns the two subtrees rooted at the lower and upper children of the root node of tree $\mathbb{T}$;
- minleafsize($\mathbb{T}, \mathbf{X}, \mathbf{y}$) returns minimum number of points contained in any leaf of tree $\mathbb{T}$ given data $\mathbf{X}$ and $\mathbf{y}$;
- loss($\mathbb{T}, \mathbf{X}, \mathbf{y}$) is a function that evaluates the objective cost of the tree $\mathbb{T}$ on data $\mathbf{X}$ and $\mathbf{y}$ after the leaf predictions of $\mathbb{T}$ have been optimized to fit $\mathbf{X}$ and $\mathbf{y}$.

Algorithm 2.1 LocalSearch

Input: Starting decision tree $\mathbb{T}$; training data $\mathbf{X}$, $\mathbf{y}$
Output: Locally optimal decision tree

```
1: repeat
2:     $\text{error}_{\text{prev}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$
3:     for all $t \in \text{shuffle}(\text{nodes}(\mathbb{T}))$ do
4:         $\mathcal{I} \leftarrow \{i : \mathbf{x}_i$ is assigned by $\mathbb{T}$ to a leaf contained in $\mathbb{T}_t\}$
5:         $\mathbb{T}_t \leftarrow \text{OptimizeNodeParallel}(\mathbb{T}_t, \mathbf{X}_{\mathcal{I}}, \mathbf{y}_{\mathcal{I}})$
6:         replace $t$th node in $\mathbb{T}$ with $\mathbb{T}_t$
7:         $\text{error}_{\text{cur}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$
8: until $\text{error}_{\text{prev}} = \text{error}_{\text{cur}}$        ▷ no further improvement possible
9: return $\mathbb{T}$
```

For Optimal Classification Trees, the loss function corresponds to the objective of Problem (2.24), which we can write as:

$$\text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y}) = \frac{1}{\hat{L}} L(\mathbb{T}, \mathbf{X}, \mathbf{y}) + \alpha \cdot \text{complexity}(\mathbb{T}), \tag{2.26}$$

where L is the number of points among the data $\mathbf{X}$, $\mathbf{y}$ that are misclassified by $\mathbb{T}$ when the labels on the leaves are chosen according to the majority rule with data $\mathbf{X}$ and $\mathbf{y}$, $\hat{L}$ is the baseline error on the training data, α is the complexity parameter, and complexity($\mathbb{T}$) is the number of splits in $\mathbb{T}$.

The approach we take considers changing one node in the current solution at a time. To do this, we simply re-optimize the split at this node so that it is locally optimal. We loop through the nodes in a random order, re-optimizing the split at each node, until we have looped through every node in the tree without making any improvement. This tree is now a local minimum for Problem (2.25) with respect to our search procedure, so we stop and begin the procedure again with a new tree. This procedure is detailed in Algorithm 2.1, which depends on the supplementary functions given in Algorithms 2.2 and 2.3.

The local search procedure given in Algorithm 2.1 depends on the function OptimizeNodeParallel in Algorithm 2.2, which re-optimizes the root node of any subtree $\mathbb{T}$ to be locally optimal for the given training data $\mathbf{X}$ and $\mathbf{y}$. There are three possible actions we take in this procedure to improve the root node. First, we can

Algorithm 2.2 OptimizeNodeParallel

```
Input: Subtree $\mathbb{T}$ to optimize; training data $\mathbf{X}$, $\mathbf{y}$
Output: Subtree $\mathbb{T}$ with optimized parallel split at root
 1: if $\mathbb{T}$ is a branch then
 2:     $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}} \leftarrow \text{children}(\mathbb{T})$
 3: else
 4:     $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}} \leftarrow$ new leaf nodes
 5: $\text{error}_{\text{best}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$          ▷ error of current root
 6:
 7: $\mathbb{T}_{\text{para}}, \text{error}_{\text{para}} \leftarrow \textsc{BestParallelSplit}(\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}, \mathbf{X}, \mathbf{y})$
 8: if $\text{error}_{\text{para}} < \text{error}_{\text{best}}$ then
 9:     $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{para}}, \text{error}_{\text{para}}$          ▷ replace with parallel split
10: $\text{error}_{\text{lower}} \leftarrow \text{loss}(\mathbb{T}_{\text{lower}}, \mathbf{X}, \mathbf{y})$
11: if $\text{error}_{\text{lower}} < \text{error}_{\text{best}}$ then
12:     $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{lower}}, \text{error}_{\text{lower}}$          ▷ replace with lower child
13: $\text{error}_{\text{upper}} \leftarrow \text{loss}(\mathbb{T}_{\text{upper}}, \mathbf{X}, \mathbf{y})$
14: if $\text{error}_{\text{upper}} < \text{error}_{\text{best}}$ then
15:     $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{upper}}, \text{error}_{\text{upper}}$          ▷ replace with upper child
16: return $\mathbb{T}$
```

consider replacing the current root node with a branch node that is locally optimal for the given training data, which is found using the BestParallelSplit function. The other options we consider are to delete the current node if it is a branch node, and instead use either the lower or upper child in its place. We calculate the new loss after performing each of these three subsitutions at the root node, and perform the best of these if the resulting loss is lower than the current loss, otherwise we make no changes to the root.

The function BestParallelSplit in Algorithm 2.3 finds the locally-optimal parallel split to use at any root branch node, given the structures of the subtrees in the lower and upper children are fixed. To do this, we follow a procedure similar in nature to that used in the CART algorithm to exhaustively search all possible split locations. In each dimension of the data, we sort the points and consider placing the parallel split between each successive pair of points. For each split we consider, we evaluate the loss of tree $\mathbb{T}$ with this split at the root and all other splits unchanged. We take the split with the best such loss, which is thus a local optimum for the loss function, since we considered every possible distinct parallel split.

Algorithm 2.3 BestParallelSplit

```
Input: Lower and upper subtrees $\mathbb{T}_{\text{lower}}$ and $\mathbb{T}_{\text{upper}}$ to use as children of new split;
    training data $\mathbf{X}$, $\mathbf{y}$
Output: Subtree with best parallel split at root; error of best tree
 1: $n, p \leftarrow$ size of $\mathbf{X}$
 2: $\text{error}_{\text{best}} \leftarrow \infty$
 3: for $j = 1, \ldots, p$ do                                    ▷ loop over all dimensions
 4:     values $\leftarrow \{X_{ij} : i = 1, \ldots, n\}$
 5:     sort values in ascending order
 6:     for $i = 1, \ldots, n-1$ do                              ▷ loop over all split placements
 7:         $b \leftarrow \frac{1}{2}(\text{values}_i + \text{values}_{i+1})$
 8:         $\mathbb{T} \leftarrow$ branch node $\mathbf{e}_j^\mathsf{T}\mathbf{x} < b$ with children $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}$
 9:         if $\text{minleafsize}(\mathbb{T}) \geq N_{\min}$ then                 ▷ check feasibility
10:             error $\leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$
11:             if error $< \text{error}_{\text{best}}$ then                  ▷ save split if better
12:                 $\text{error}_{\text{best}} \leftarrow$ error
13:                 $\mathbb{T}_{\text{best}} \leftarrow \mathbb{T}$
14: return $\mathbb{T}_{\text{best}}, \text{error}_{\text{best}}$
```

As mentioned earlier, we repeat the local search procedure for a number of starting trees, and take the tree with the best loss as the final solution. This allows us to better explore the search space for the global minimum and avoid getting trapped in local minima. For the starting trees, we generate trees using a similar method as for Random Forests [29]. Namely, the trees are generated as normal for CART, except at each stage of the tree growing process we are only allowed to consider a random subset of the features to split on. We have found that $\sqrt{p}$ is a good size for this random subset, similar to Random Forests. Note that unlike Random Forests, we do not take bootstrap sample of the training points to ensure that the trees we generate satisfy the minimum leaf size when applied to the original training set, and thus are feasible starting points.

This approach using Random Forests to generate starting points works well, since this produces trees that are of high quality individually, yet there is still significant variance across the set of trees generated from this process. CART, as a deterministic algorithm, would not be well-suited for this task.

We conclude by observing that the local search for each starting solution is independent of all other starting points, and so the search process is trivially parallelizable across the starting solutions. This means the speed increases approximately linearly with the number of cores available, allowing for very large speedups due to parallelization.

Complexity analysis of local search

We will now analyze the complexity of the local search procedure and show that it runs efficiently even in the worst case, and is not significantly more expensive than classical heuristics for parallel decision tree construction like CART.

First, we observe that we require the points in sorted order for each feature in the BestParallelSplit function in Algorithm 2.3. This order does not change for the duration of the local search procedure, so we can simply presort the data in each of the p features for a cost of $O(np \log n)$ as the first step in the algorithm.

We now analyze the cost of BestParallelSplit. For each feature of the data, we consider placing the split between each pair of points and evaluating the misclassification error for each split. However, we do not need to calculate the error from scratch at each split position, assuming that the error of first tree considered is available. We search the splits in sorted order, and so we can update the error incrementally by simply switching which branch the next point is assigned to and updating the misclassification counts accordingly. This means that calculating the error inside the loop of BestParallelSplit has cost of just $O(K)$ using the majority rule, thus the total cost of the loop is $O(npK)$ for evaluating the error of each split location between n points in each of the p features.

Next, we consider the OptimizeNodeParallel function in Algorithm 2.2. This first calculates the errors for the current, lower and upper subtrees, followed by running BestParallelSplit. Observe that the upper subtree is the same as the first tree considered inside BestParallelSplit, so we can pass the error of this upper tree into the function and avoid recalculating it as we assumed, meaning BestParallelSplit has a total cost of $O(npK)$. We will also assume for now that the errors of the current, lower and upper subtrees are all provided to OptimizeNodeParallel, so the total cost of this function is $O(npK)$.

Finally, we consider the cost of the overall local search function in Algorithm 2.1. Inside the innermost loop, we calculate the assignment of points to leaves in the tree, which takes $O(nT)$ time, where T is the number of nodes in the tree, since each point can have $O(T)$ tests when determining which leaf contains it. We then run OptimizeNodeParallel for cost $O(npK)$ and update the tree $\mathbb{T}$ and current error at cost $O(T)$. We also assumed the error of the current, lower and upper subtrees were made available to OptimizeNodeParallel, which can be obtained by calculating the assignment of points to leaves for each of the subtrees, again in $O(nT)$ time, and then calculating the resulting misclassification cost using the majority rule at cost $O(KT)$. This means the total cost for a single iteration of this loop is $O(npK + nT + KT) = O(npK + nT)$ under the assumption that $n \gg K$.

At each iteration of the local search, we calculate the error once for cost $O(nT + KT)$ before running this loop over each node in the tree, which gives an iteration cost of $O(nT + KT) + T \cdot O(npK + nT) = O(npKT + nT^2)$.

However, we can improve the runtime of this loop as follows. Suppose that we exit the loop after the first improvement we make, and then start the next local search iteration from scratch. This means that the tree is unchanged for the duration of the local search iteration, and so we can precompute the errors and point-to-leaf assignments of all subtrees in the tree $\mathbb{T}$ before the loop, at a cost of $O(nT)$ for the assignments, and $O(KT)$ for each of the $O(T)$ error calculations, for a total of $O(nT + KT^2)$. This reduces the cost of the inner loop iteration to just $O(npK)$, making the total cost of a local search iteration $O(nT + KT^2) + T \cdot O(npK) = O(npKT)$, since the number of nodes in the tree is at most $O(n)$ with one point per leaf. This optimization has therefore removed the second term from the runtime.

If there is no penalty on complexity, $\alpha = 0$, we can bound the number of local search iterations. In this case, the error is just the misclassification score which is integral, non-negative, and bounded above by n. Each iteration strictly decreases the error, so this implies at most n iterations can occur before termination. This means the total cost of the local search procedure for a single starting solution is $O(n^2pKT)$. We conduct this for a constant number R of random restarts, which does not change the complexity of the local search procedure.

The cost of generating a random starting tree is the same as that of generating a CART tree, which is $O(npKT)$: CART uses a function similar to BestParallelSplit as the innermost unit of work at a cost of $O(npK)$. This is run at each node in the tree, for a total cost of $O(npKT)$.

The overall cost of the OCT algorithm is therefore $O(np\log n)$ for the presorting, $O(npKT)$ for generating the starting solutions, and $O(n^2pKT)$ for the local search. This final step dominates giving a total cost of $O(n^2pKT)$. The cost of CART is just $O(np\log n)$ for the presorting and $O(npKT)$ for generating the tree for a total cost of $O(np(KT+\log n))$

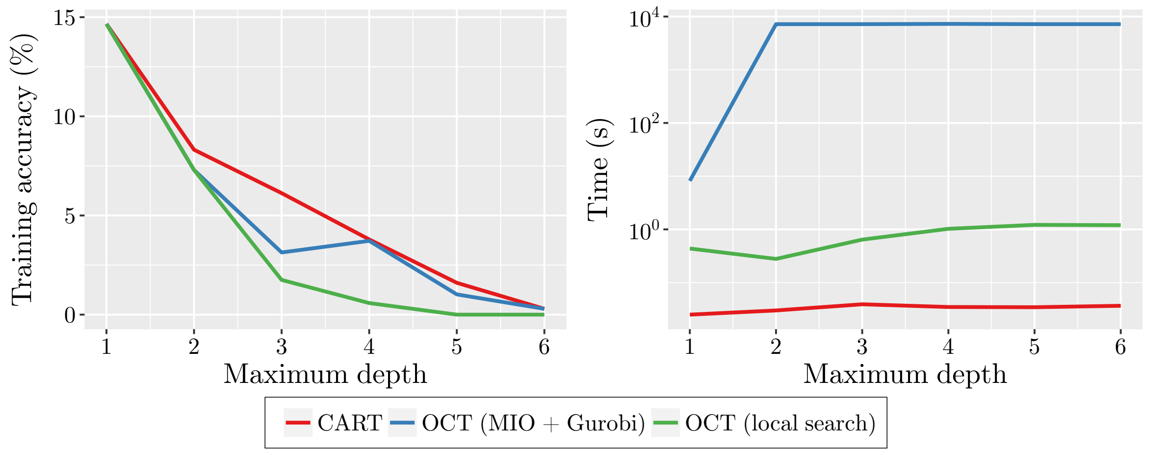
In the absolute worst case, each split in the tree would separate a single point only, and so the number of nodes T would be $O(n)$, giving a worst-case cost of $O(n^3pK)$. Under this worst case, the cost of CART would be $O(n^2pK)$, and so the additional power of OCT increases the cost simply by a factor of n.

A set of more realistic assumptions for practical use is that the number of nodes in the tree would be $O(\log n)$ and the number of local search iterations required is also $O(\log n)$. Under this scenario, the cost of OCT is $O(npK\log^2 n)$ and the cost of CART is $O(npK\log n)$, and so OCT is only more expensive by a factor of $\log n$. Empirically we have found that the number of local search iterations required remains almost constant as the number of points increases, and therefore assuming the number of iterations to be $O(\log n)$ is realistic and perhaps even conservative.

Does the local search procedure work?

We demonstrate the speed and effectiveness of this local-search approach by revisiting the earlier example that used the "Banknote Authetication" dataset. Figure 2-6 shows the results when we use the local-search approach to solve the OCT problem with 100 random restarts. We immediately see that the local-search approach generates solutions of much higher quality than both CART and the MIO-based OCT at the higher depths, and even reaches 0% error at depth 6. At depths 3 and higher, there is clear evidence that the solutions generated by the MIO approach were not optimal,

Figure 2-6: Training accuracy (%) and running time for each method on "Banknote Authentication" dataset.



as there is clearly great room for improvement. Finally, it takes around one second in the largest cases to construct and optimize all 100 of the random restarts using the local search, so the method runs in times within 1–2 orders of magnitude of CART, and is around 4 orders of magnitude faster than the MIO approach, which makes it a much more powerful and practical alternative to CART than the MIO approach. Finally, we note that these runtimes were measured using just a single CPU core. Since the performance increases linearly with the number of cores, running the same experiments on a typical machine with 16 cores available would give runtimes approximately the same as CART.

2.4 A Method for Tuning Hyperparameters

In this section, we present an approach for tuning the hyperparameters in the Optimal Classification Trees problem that we have found empirically to give strong performance out-of-sample.

The hyperparameters in the OCT problem are the complexity parameter α, the maximum depth of the tree D, and the minimum leaf size $N_{\min}$. Varying any or all of these parameters changes the degree to which the tree is optimized to fit the training data:

- The complexity parameter controls the tradeoff between the training error and the number of splits in the tree, ensuring that increasing the number of splits in the tree leads to a corresponding reduction in training error that is sufficiently large;
- The maximum depth controls the number of sequential splits that can be used to classify any point, to make sure the decision rules are not too complicated;
- The minimum leaf size controls the degree to which the tree can be fit to smaller groups of training points; increasing this parameter leads the tree to focus on the training points as clusters rather than the individual points.

In order to create a tree that performs well out-of-sample, we need to ensure that we do not overfit to the training data, which we can achieve by tuning the values of the hyperparameters to arrive at the "right-sized tree". This is typically conducted by reserving some of the training data as a validation set, and using the performance of the trained model on the validation set to guide the choice of hyperparameter values. In this sense, the performance on the validation set is used as a proxy for the true out-of-sample performance, and we expect the hyperparameters that give the best performance on the validation set to also perform well on new data.

A standard approach to tuning hyperparameters is to use an exhaustive grid-search, where every possible combination of the hyperparameters is used to train a model on the training set. We then evaluate each model on the validation set and choose the combination of hyperparameters that gave the model with the best performance. This approach works well for hyperparameters that take discrete values over a small range, such as the maximum depth D, which is typically from 1–15. However, if the hyperparameter has a large range or is continuous (like the complexity parameter), it is not feasible (or perhaps even possible) to search every possible value. One potential remedy is to discretize the range of possible values and select the best from this set. The quality of a tuned value is therefore likely to depend on the granularity of the discretization, with more precise values requiring a finer mesh of values.

Fortunately, despite the complexity parameter being continuous-valued, it has a discrete effect on the tree. Namely, varying the complexity parameter with all other hyperparameters fixed changes the number of nodes in the tree; the maximum number of nodes is achieved with $\alpha = 0$, and increasing α from here will reduce the number of nodes in the tree. For each tree, there is a corresponding interval for α from which any value will generate this tree. We can therefore discretize the range of α intelligently using these intervals and only test the "critical" values of α, avoiding any wasted effort in testing values for α that would give trees we have already found.

This intelligent approach to discretization is the basis for the *cost-complexity pruning* used by CART [25]. This procedure first generates a tree with $\alpha = 0$, then prunes the splits of the tree one-by-one in order of strength to generate a decreasing sequence of trees. Next, the tree with the best performance on the validation set is identified, and the range of α values that would give rise to this tree is determined. Finally, the midpoint of this interval is selected as the tuned value for α.

We will now reproduce the details of the cost-complexity pruning algorithm. At each step in the pruning process, we need to identify the weakest of the branch nodes in the tree to prune and replace with a leaf node. Given a tree $\mathbb{T}$, we let $R(\mathbb{T}_t)$ be the misclassification error of the tth branch node, complexity$(\mathbb{T}_t)$ be the number of splits in the subtree rooted at the tth branch node, and $R(t)$ be the misclassification error that would be realized if the tth branch node was replaced with a leaf. The change in objective function by replacing the tth branch node with a leaf is therefore

$$\underbrace{R(t)}_{\text{leaf error}} - \underbrace{\left[R(\mathbb{T}_t) + \alpha \cdot \text{complexity}(\mathbb{T}_t)\right]}_{\text{branch error}}. \tag{2.27}$$

This substitution will improve the objective if (2.27) is negative, or equivalently if

$$\alpha > \frac{R(t) - R(\mathbb{T}_t)}{\text{complexity}(\mathbb{T}_t)} \triangleq g(t). \tag{2.28}$$

The weakest branch node in the tree will be the one with the smallest value of $g(t)$, as this is the first node that will be replaced as we increase the complexity penalty

Algorithm 2.4 Prune

```
Input: Starting decision tree $\mathbb{T}$
Output: Vector $\boldsymbol{\alpha}$ of critical values for complexity parameter; vector $\mathbf{v}$ of corresponding validation errors for each critical value
 1: $\alpha_1 \leftarrow 0$
 2: $v_1 \leftarrow V(\mathbb{T})$
 3: $i \leftarrow 1$
 4: while complexity($\mathbb{T}$) $> 0$ do
 5:     $i \leftarrow i + 1$
 6:     $\alpha_i \leftarrow \infty$
 7:     for all $t \in$ branches($\mathbb{T}$) do
 8:         $a \leftarrow \frac{R(t)-R(\mathbb{T}_t)}{\text{complexity}(\mathbb{T}_t)}$
 9:         if $a < \alpha_i$ then
10:             $\hat{t} \leftarrow t$
11:             $\alpha_i \leftarrow a$
12:     replace branch node $\hat{t}$ in $\mathbb{T}$ with leaf node
13:     $v_i \leftarrow V(\mathbb{T})$
14: return $\boldsymbol{\alpha}, \mathbf{v}$
```

Algorithm 2.5 TuneCP

```
Input: Starting decision tree $\mathbb{T}$
Output: Tuned value of $\alpha$ that minimizes validation error
 1: $\boldsymbol{\alpha}, \mathbf{v} \leftarrow$ Prune($\mathbb{T}$)           ▷ Generate sequence of pruned trees
 2: $i \leftarrow \arg\min_j v_j$                       ▷ Find interval with minimal error
 3: return $\frac{1}{2}(\alpha_i + \alpha_{i+1})$                ▷ Return midpoint of interval
```

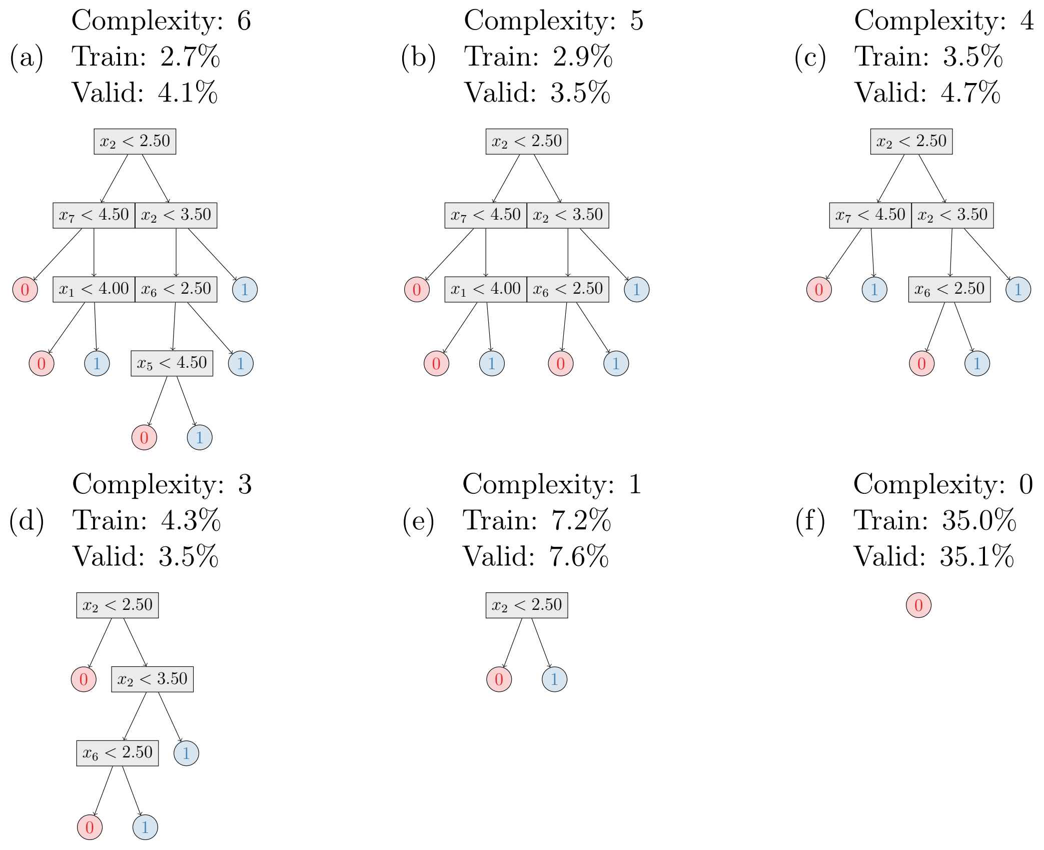
α. We therefore identity $\hat{t} = \arg\min_t g(t)$ and replace this node with a leaf, and then continue pruning from the beginning. At each step in the process, we track the validation error of the current tree $\mathbb{T}$, denoted by the function $V(\mathbb{T})$, and the current critical value of α. This procedure is detailed in Algorithm 2.4.

Finally, we identify the tree with the smallest validation error, and use the corresponding critical values of α to determine the final tuned value. Algorithm 2.5 describes this process.

Figure 2-7 demonstrates the cost-complexity pruning process as applied to a tree trained on the "Breast Cancer Wisconsin" dataset from the UCI Machine Learning Repository [77].

We might simply use the same cost-complexity pruning approach to tune the value

Figure 2-7: Example of pruning process following Algorithm 2.4 on the "Breast Cancer Wisconsin" dataset. The starting tree in Figure 2-7a has the largest complexity and lowest training error. In each step of the pruning process, the weakest split in the tree is identified using Equation (2.28) and then replaced with a leaf. Pruning this split causes the complexity to drop and the training error to increase. Eventually we arrive at a tree that is just a single root node. For each tree in this pruning sequence we also calculate the corresponding error on the validation set. Our goal is to choose the complexity parameter α that induces the right-sized tree, which is traditionally done by choosing the α that would have led to the tree with the smallest validation error, which is demonstrated by the process in Algorithm 2.5.



of α in the OCT model, but we have found empirically that this method does not lead to tuned values that perform well on new data. To demonstrate this, Figure 2-8 shows results of applying cost-complexity pruning to OCT trees on the same "Breast Cancer Wisconsin" dataset from Figure 2-7. Both plots correspond to OCT trees from different starting solutions that had 100% in-sample accuracy when trained with $\alpha = 0$. These plots show the validation error of the pruned tree as a function of the complexity parameter α, with each horizontal line in the plot corresponding to one of the trees in the pruning sequence, and the endpoints of this line identifying the range of values for α that lead to this tree. In order to determine the tuned value of α, we want to find the value that minimizes the validation error on these plots.

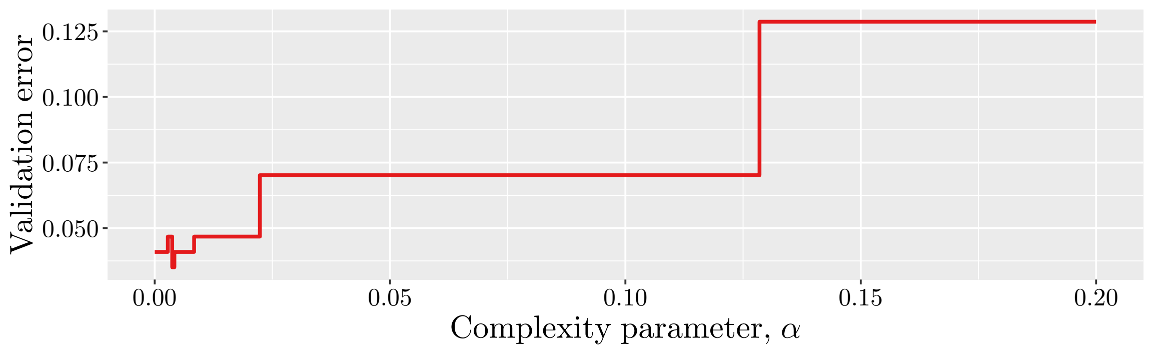
Comparing the pruning curves in Figures 2-8a and 2-8b, we immediately see that the curves are significantly different. Despite both OCT trees having the same perfect in-sample accuracy, the value of α that minimizes the validation error is different for each tree, and in fact there is no overlap between the minimizing values of α for each tree. There are often many possible high-quality solutions that can arise when training the model with $\alpha = 0$, because the complexity of the tree is not penalized. We could get significantly different values for the tuned value of α depending on which of these optimal trees we select as the tree for pruning. This makes the cost-complexity pruning an unstable procedure with high variance in the tuned values of α.

If we consider the curve for the tree in Figure 2-8b, we see there are two intervals in which the validation error reaches its minimum, raising the question of which of these intervals we should select. Another problem is that the validation error is not very smooth as a function of α, and even after identifying the interval that minimizes the validation error, we still have to select a single value as the final answer. Traditionally the midpoint of the interval is taken to be the final value, however there is little indication that choosing the midpoint is the best choice for the true minimizer, especially if the interval is large.

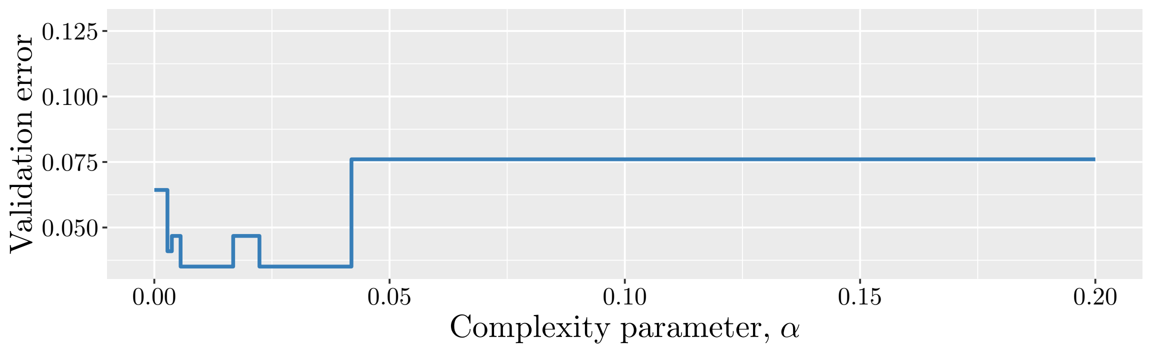
These problems in combination lead to estimates for the optimal value of α that have inherently high variance, due both to the uncertainty in which tree will be selected as the basis for pruning, and also the lack of granularity when selecting the

Figure 2-8: An example showing the instability of cost-complexity pruning with two OCT trees trained on the "Breast Cancer Wisconsin" dataset. Both trees have 100% in-sample accuracy when the complexity parameter is zero. The plots show the pruning curve generated by each tree using cost-complexity pruning. Despite having the same perfect in-sample accuracy, the pruning curves are significantly different and give different predictions for the value of the complexity parameter that minimizes the validation error.

(a) 1st OCT tree



(b) 2nd OCT tree



α with the best validation performance. As a result, a procedure that generates trees having used this approach to tune the value of α will have high variance and thus the trees it generates will have reduced ability to perform out-of-sample.

The instability in the cost-complexity pruning procedure for CART was observed by Breiman [27], alongside many other unstable methods. He proposed an approach for stabilizing such unstable procedures in the context of best-subset selection by perturbing the data many times and averaging the results of the models fit to each perturbed dataset. Similar approaches were later applied in the context of decision trees to give *Bagging* [26], *Arcing* [28] (now known as boosting), and *Random*

Forests [29]. However, these approaches achieved stability by averaging the results of many decision trees rather than providing a stable approach for producing a single decision tree. As Breiman [27] noted:

> *An interesting research issue we are exploring is whether there is a more stable single-tree version of CART.*

To resolve this instability in decision tree training, we now present *batch cost-complexity pruning*, an approach that reduces the variance in the tuning process for OCT. This approach resolves the question of which tree to use as the basis for pruning by selecting *all* of the high-quality trees generated in the local search, and in doing so, smoothens the estimates of validation error as a function of α, enabling us to obtain a higher-precision estimate for the value of α that maximizes performance on the validation set.

The key idea behind the batch cost-complexity pruning method is using multiple trees together to conduct the pruning procedure. From the trees generated during the local search process, we select the k trees with the best training error (empirically we have found setting $k = n/10$ gives the best results, i.e., selecting the best 10% of the trees). For each of the trees in this subset, we conduct the standard cost-complexity pruning process, using PRUNE from Algorithm 2.4, and construct the curve of validation error as a function of complexity parameter:

$$f_j(\alpha) = \sum_{i=1}^{|\boldsymbol{\alpha}^j|} (v_i^j - v_{i-1}^j)\mathbb{1}\{\alpha \geq \alpha_i^j\}, \tag{2.29}$$

where $\boldsymbol{\alpha}^j$ and $\mathbf{v}^j$ are the results of the PRUNE function on the jth tree, and we take $v_0^j = 0$ for convenience of notation.

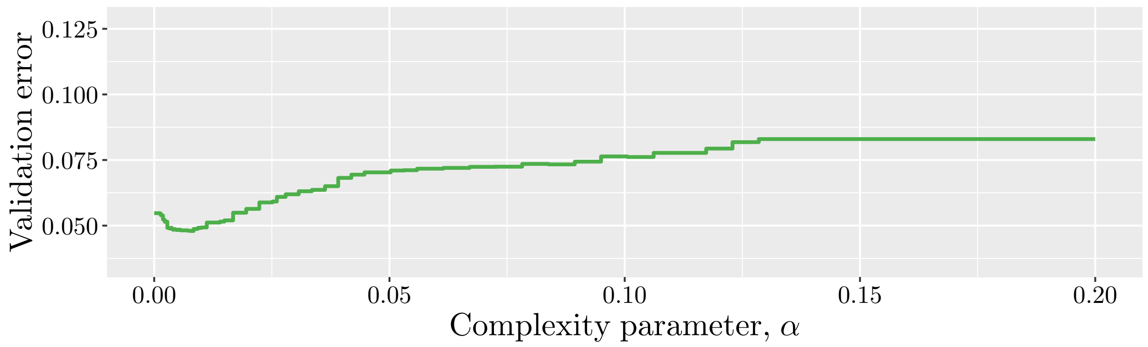
We then average the curves of all k trees to obtain a single smoothed curve of mean validation error as a function of complexity parameter:

$$f(\alpha) = \frac{1}{k}\sum_{j=1}^{k} f_j(\alpha). \tag{2.30}$$

Algorithm 2.6 BatchTuneCP

```
Input: Vector trees of solutions from local search; batch size $k$
Output: Best validation error; tuned value of complexity parameter
1: Sort trees by training error
2: for $j = 1, \dots, k$ do
3:     $\boldsymbol{\alpha}^j, \mathbf{v}^j \leftarrow \textsc{Prune}(trees_j)$
4:     $f_j(\alpha) \leftarrow \sum_{i=1}^{|\boldsymbol{\alpha}^j|} (v_i^j - v_{i-1}^j) \mathbb{1}\{\alpha \geq \alpha_i^j\}$
5: $f(\alpha) \leftarrow \frac{1}{k} \sum_{j=1}^{k} f_j(\alpha)$
6: $v_{\text{best}} \leftarrow \min_\alpha f(\alpha)$
7: $\mathcal{A} \leftarrow \{\hat{\alpha} : \hat{\alpha} \in \arg\min_\alpha f(\alpha)\}$          ▷ All function minimizers
8: $\alpha_{\text{best}} \leftarrow \frac{1}{2}(\min_{\alpha \in \mathcal{A}} \alpha + \max_{\alpha \in \mathcal{A}} \alpha)$          ▷ Midpoint of minimizers
9: return $v_{\text{best}}, \alpha_{\text{best}}$
```

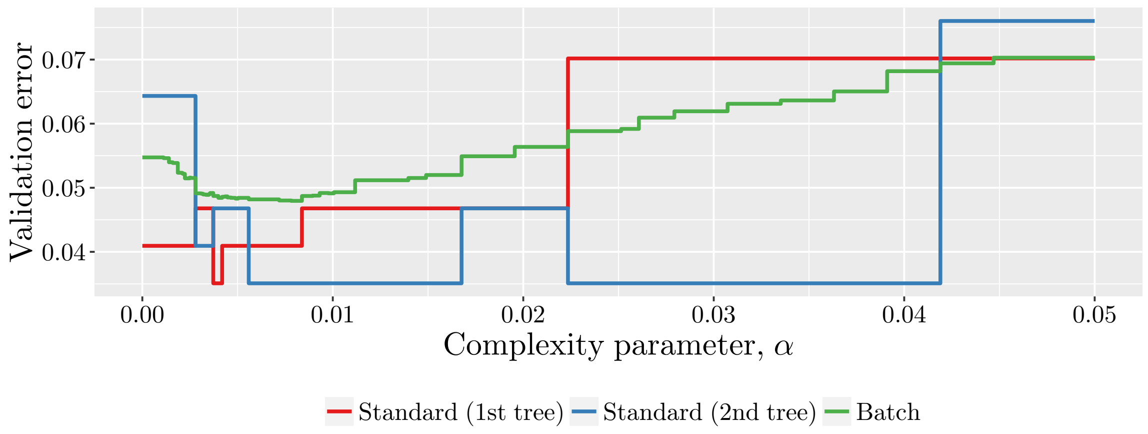
Figure 2-9: Tuning results using batch pruning on "Breast Cancer Wisconsin" dataset.



We then proceed to identify the minimizing range of α values for this function and use the midpoint of this range as the final tuned value in the same way as for standard cost-complexity pruning. The full algorithm is given in Algorithm 2.6.

We will now illustrate the advantages of this algorithm on the same example as before. Figure 2-9 shows the batch pruning curve for the same "Breast Cancer Wisconsin" dataset as in Figure 2-8. We immediately see that the curve is a lot smoother and has a more obvious minimizing value. To make this comparison clearer, Figure 2-10 compares the curves generated by the standard and batch pruning methods in the region where the curves are minimized. We can see that the batch pruning curve has a well-defined minimizing point around 0.008, whereas the standard pruning curve indicates the minimizer would be around 0.004 if using the first tree, and anywhere from 0.006 to 0.042 for the second tree. This example shows the power of the batch

Figure 2-10: Comparison of pruning methods on “Breast Cancer Wisconsin” dataset in the region where validation error is minimized.



Algorithm 2.7 Tune

Input: Maximum depth of tree D_{max}; batch size k

Output: Tuned value for complexity parameter; tuned value for depth

```
1: $v_{\text{best}} \leftarrow \infty$
2: for $D = 1, \ldots, D_{\text{max}}$ do
3:     $\mathbb{T} \leftarrow$ decision tree trained with maximum depth $D$
4:     $\alpha, v \leftarrow$ BatchTuneCP$(\mathbb{T}, k)$
5:     if $v < v_{\text{best}}$ then
6:         $v_{\text{best}} \leftarrow v$
7:         $\alpha_{\text{best}} \leftarrow \alpha$
8:         $D_{\text{best}} \leftarrow D$
9: return $\alpha_{\text{best}}, D_{\text{best}}$
```

pruning process in reducing the uncertainty in the pruning process. By combining the pruning of multiple high-quality trees, we are able to zero in on a very precise estimate for the optimal value of α.

A procedure for tuning all hyperparameters

The batch cost-complexity pruning procedure that we have presented is a powerful method for tuning the value of α, and we will now present a method for tuning all of the hyperparameters in the OCT model that takes advantage of this batch process. This method simply embeds the batch pruning inside an exhaustive grid search over the remaining parameters. The full procedure is given in Algorithm 2.7.

In Algorithm 2.7, we only tune the complexity parameter α and the depth D. We have found empirically that there is little advantage to be gained in tuning the minimum leaf size $N_{\min}$; the out-of-sample performance is largely the same if we simply fix $N_{\min} = 1$, and so it is unlikely that tuning this parameter will be worth the additional cost. However, it is trivial to modify Algorithm 2.7 to tune $N_{\min}$ by adding another outer loop over the candidate values for this parameter.

As mentioned earlier, we have found empirically that $k = n/10$ is a good batch size to use for this validation process, and leads to tuned values that perform well out-of-sample across a wide range of problems.

2.5 Experiments with Synthetic Datasets

In this section, we examine the performance of Optimal Classification Trees on a variety of synthetically-generated datasets in order to understand how effective the method is at discovering the underlying ground truth in a dataset whose structure is in fact described by a decision tree.

Experimental Setup

The experiments we conduct are adaptations of the experiments by Murthy and Salzberg [90]. These experiments use a single decision tree as the ground truth to generate datasets, and then different methods are used to induce decision trees on these datasets and are then compared to the ground truth tree to evaluate performance. To construct the ground truth, a decision tree of a specified depth is first created by choosing splits at random. The leaves of the resulting tree are then labeled such that no two leaves sharing a parent have the same label, otherwise this parent node could simply be replaced with a leaf of the same label without affecting the predictions of the overall tree.

The training and test datasets are created by generating each $\mathbf{x}_i$ as a uniform random vector, and then using the ground truth tree to assign the corresponding label to the point. The size of the training set is specified by the experiment, and the size of the test set is always $2^{D_{true}} \cdot 2000$, where D_{true} is the depth of the ground truth tree as specified by the particular experiment. 25% of the training set is reserved for validation purposes in order to tune the complexity parameter α and the maximum depth D using the procedure in Algorithm 2.7.

In order to determine the ability of each method to learn the true model described by the data, we record three measures of tree quality in all experiments:

- Out-of-sample accuracy: accuracy on the test set.

- True discovery rate (TDR): the proportion of splits in the ground truth tree that are also present in the generated tree. This measures the ability of the method to recover the true splits from the data.

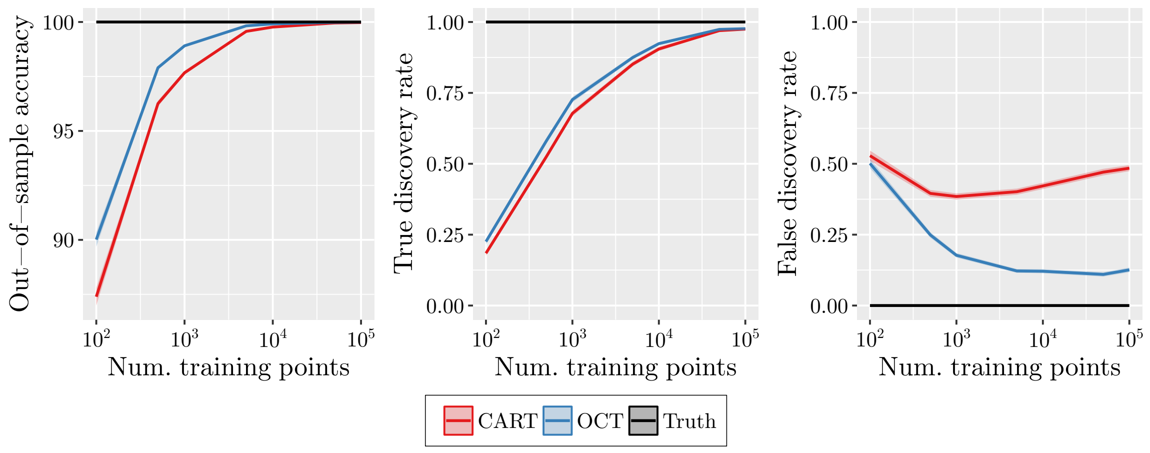
- False discovery rate (FDR): the proportion of splits in the generated tree that are not present in the ground truth tree. This measures the amount of noise in the generated tree.

For each experiment, 400 random trees are generated as different ground truths, and training and test set pairs are created using each tree. The quality measures are calculated over all 400 instances of the problem, and in the figures we present the means of each measure with the standard error indicated by the error ribbon around each line.

In these experiments we compare our OCT method to the standard CART heuristic. Note that [90] found that C4.5 and CART gave near-identical results for all experiments they conducted and reported only the C4.5 results, so our results for CART should also be similar to those that would be obtained using C4.5.

Each experiment varies one parameter while holding all others constant. Unless otherwise specified, the experiments are run with $n = 1000$ training points in $p = 5$ dimensions and $K = 2$ classes, the ground truth trees are depth $D = 4$, there is no noise added to the training data, and OCT is run with 1000 random restarts.

Figure 2-11: Synthetic experiments showing the effect of training set size.



Implementation Details

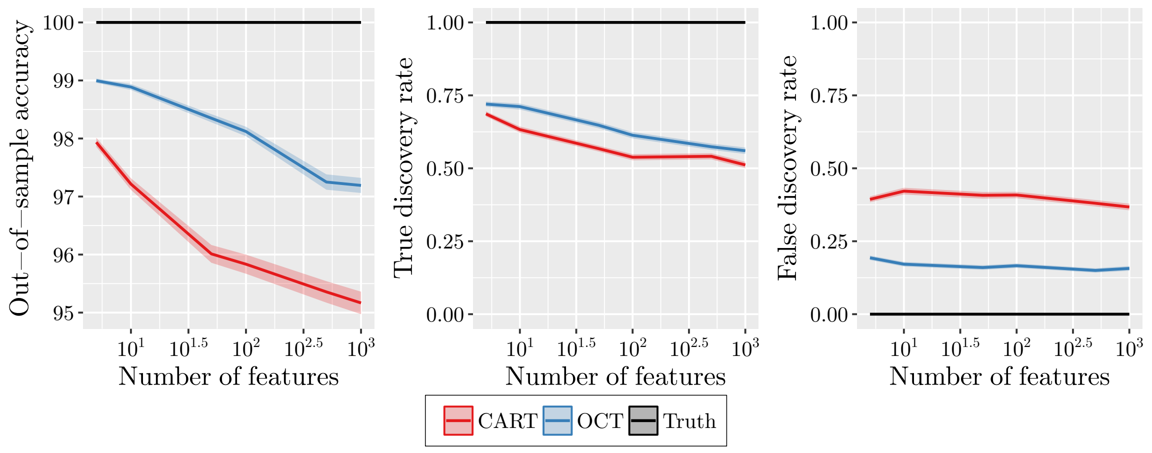
The Optimal Classification Tree method as detailed in Section 2.3 was implemented in the Julia programming language, which is a high-level, high-performance dynamic programming language for technical computing [21]. We used Algorithm 2.7 to tune the hyperparameters.

For testing the performance of CART, we used the rpart package [110] in the R programming language [106]. The hyperparameters $N_{\min}$ and α correspond to the parameters `minbucket` and `cp`. As for OCT, `minbucket` was set to 1, and the `cp` was tuned using standard cost-complexity pruning as described in Section 2.4.

The effect of the amount of training data

The first experiment investigates the effect of the amount of training data relative to the complexity of the problem. Figure 2-11 shows a summary of the results. We see that both methods increase in out-of-sample accuracy as the training size increases, approaching 100% accuracy at the higher sizes. In all cases, OCT has higher accuracy than CART on training sets of the same size. For smaller training sets, the difference is about 1–3%, and this difference shrinks as the training set size increases. This shows that OCT performs stronger in data-poor regimes, while CART is able to eventually achieve comparable accuracies if supplied enough data. This offers clear

Figure 2-12: Synthetic experiments showing the effect of number of features.

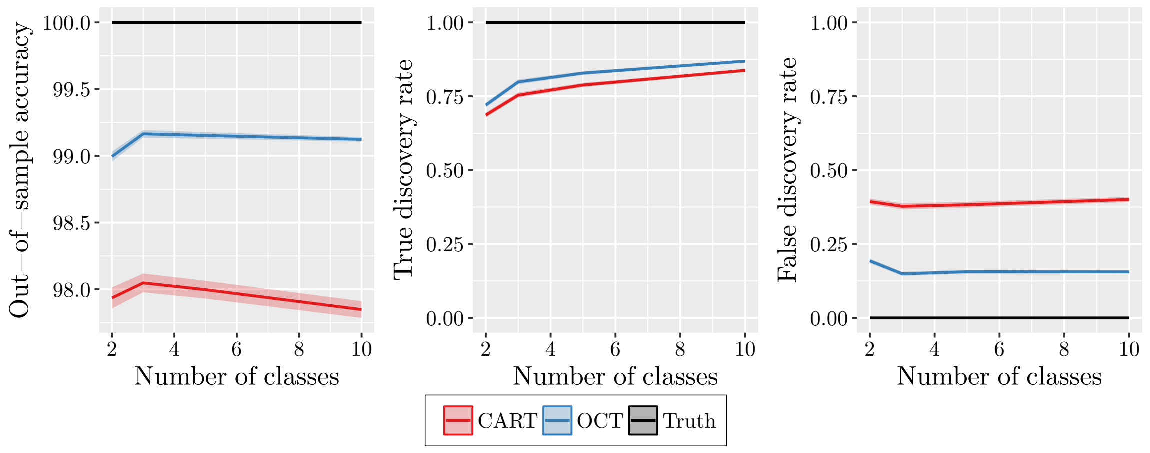


evidence against the notion that optimal methods tend to overfit the training data in data-poor scenarios; in fact the optimal method performs stronger. The TDR of both methods increases as the training size increases, indicating that it is easier to recover the truth with more training data. OCT has a small but significant advantage in TDR of 2–5% over CART at all but the largest sizes. The biggest difference between the methods is in the FDR. As the training set size increases, CART's FDR hovers between 40–50%, despite the increasing out-of-sample accuracy and TDR. This means that even though the CART trees are performing increasingly well for prediction, they are not improving their interpretability with more data, as around half of the splits in the trees generated are unrelated to the true data generation process. In stark contrast, the FDR of OCT falls significantly as the training size increases, leveling out at 10–15%. This demonstrates that OCT is much better able to identify the whole truth, and nothing but the truth, in the data.

The effect of dimensionality

The second experiment considers the effect of the dimensionality of the problem, and the results are shown in Figure 2-12. We see that both methods lose accuracy as the problem dimension increases, reinforcing our intuition that the problem is more difficult at higher dimensions. At all dimensions, OCT has significantly higher accuracy than CART, and this difference grows from 1% at low dimensions to around

Figure 2-13: Synthetic experiments showing the effect of number of classes.

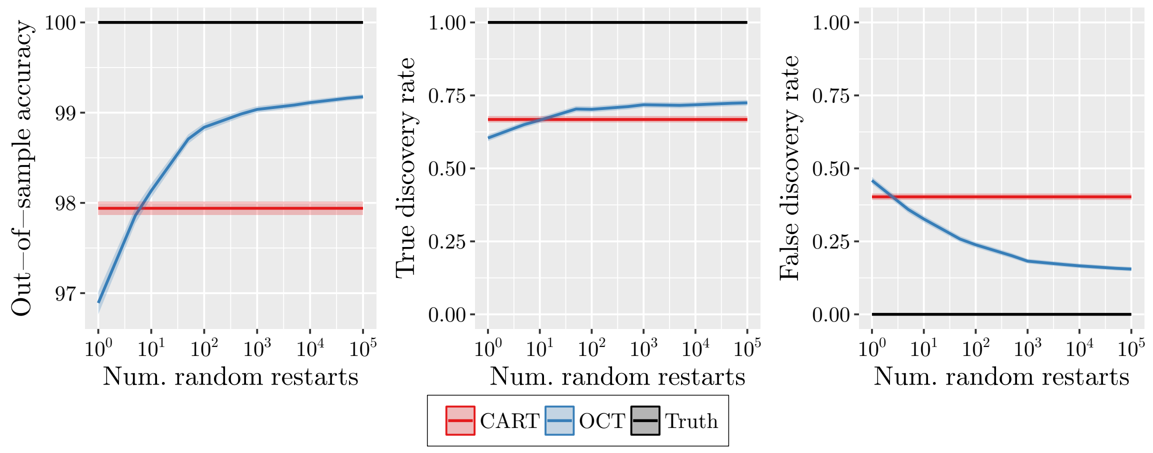


2% at higher dimensions. The TDR shows the same trend as the accuracy, decreasing as the problem difficulty increases, with OCT outperforming CART by about 3–8%. Again, there is a significant difference in the FDR of each methods. CART has an FDR around 35–40%, while OCT is less than half this at around 15%, meaning the trees generated by OCT will be much more useful for interpretation. Finally, we note that the highest dimension we tested was $p = 1000$, the same as the number of training points, $n = 1000$, showing that OCT is able to perform well in a high-dimensional setting.

The effect of the number of classes

The third experiment considers the effect of the number of classes in the classification problem on the performance of each method. Figure 2-13 shows the results of this experiment. The out-of-sample accuracy of CART decreases slightly as the number of classes increases, whereas the accuracy of OCT remains nearly constant. This is an interesting result, because it seems to indicate that the greedy approach used by CART is better suited to binary classification problems, and does not transfer perfectly to multi-class problems. OCT improves upon the TDR of CART by 3–5%, and more than halves the FDR, echoing the results of the previous experiments.

Figure 2-14: Synthetic experiments showing the effect of number of random restarts.



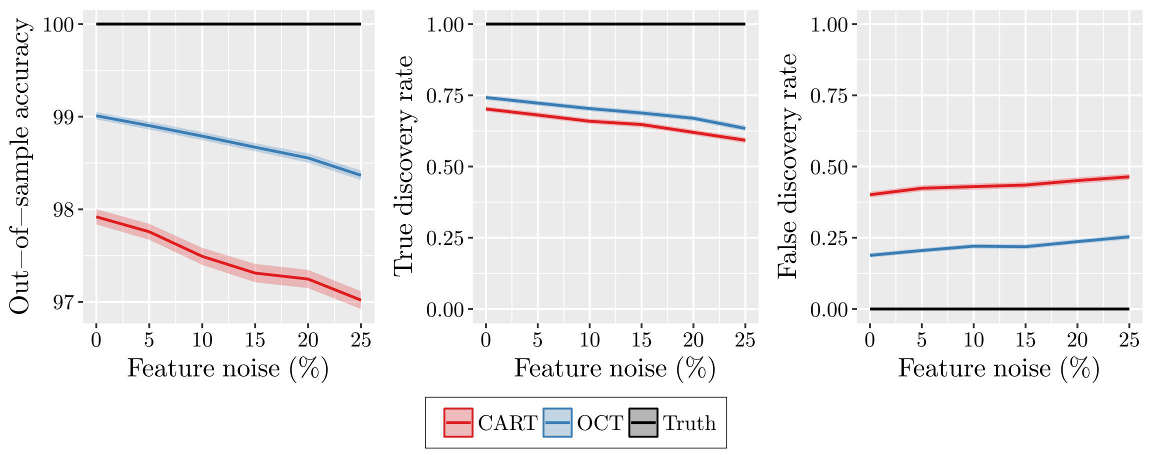
The effect of the number of random restarts

The fourth experiment measures the performance of OCT as a function of the number of random restarts used in the local search. The results are shown in Figure 2-14. We see that the number of random restarts has a large effect on the accuracy at first, but after around 100–1000 restarts the rate of improvement diminishes; the performance with 1000 and 100000 restarts is very close, with out-of-sample accuracies of 99.0% and 99.1%, respectively. The TDR is similar, as it increases initially, but eventually levels out. The trend is most apparent in the FDR, which decreases rapidly until 1000 restarts, where it is about half of CART's FDR, and then decreases more slowly with more restarts. Together, these results seem to indicate that the minimum number of restarts we should be using is around 100–1000, as this is where the rates of improvement begin to decrease. After this point, the performance still improves with additional restarts, but at a much slower rate, so it may not be as worthwhile to continue training after this point, depending on the application.

The effect of noise in the training data

The fifth experiment considers the effect of adding noise to the features of the training data. We introduce noise by selecting a random $f\%$ of the training points and to each of these adding random noise $\epsilon \sim U(-0.1, 0.1)$ to every feature, where f is the

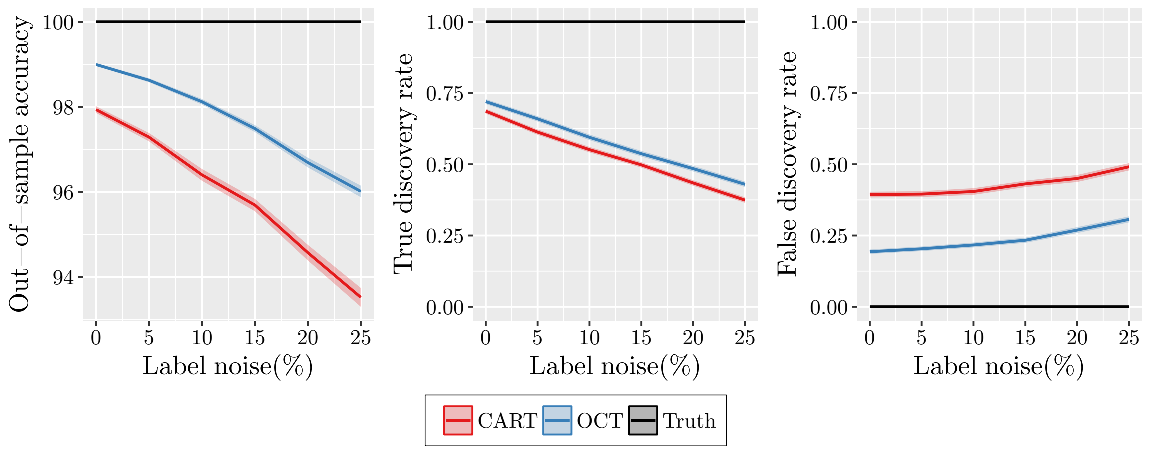
Figure 2-15: Synthetic experiments showing the effect of feature noise.



parameter of the experiment. Figure 2-15 shows the impact of increasing levels of this feature noise in the training data. As we would expect, the accuracies of each method decrease as the level of noise increases. At all levels of noise, OCT has accuracies that are 1–1.5% higher than CART. Even at the highest level of noise, where 25% of the training data are perturbed in all features by up to 10%, OCT has an out-of-sample accuracy of 98.4% and is significantly higher than CART, countering the idea that optimal methods are less robust to noise than current heuristic approaches due to problems of overfitting. Similar to the accuracy, the TDR decreases as the noise increases, but OCT is again able to better find the truth in the noise, having a 4–5% improvement in TDR. The FDR increases slightly for both methods with noise, as it becomes harder to extract only the truth from the data, but the FDR of CART is again roughly twice that of OCT.

The final experiment also examines the impact of adding noise to the training data, but this time to the labels rather than the features. Noise was added to the training set flipping the labels of a random $f\%$ of the points. Figure 2-16 shows the impact of increasing levels of label perturbations in the training data. As the level of noise increases, the accuracies of both methods decrease, similar to the previous experiment and as would be expected. The difference in accuracies is relatively constant at around 1–1.5% for all levels of noise. The TDR shows a similar decreasing trend, with OCT outperforming CART by around 5%. The FDR of both methods again

Figure 2-16: Synthetic experiments showing the effect of label noise.



increases slightly with label noise, with OCT again having an FDR of about half that of CART.

To further investigate the effect of noise on our results, we conducted each of the experiments (apart from the final two) again with 5% feature and 5% label noise added to the training data. In every case, we found that the results of the experiments were nearly identical to the results without noise, apart from some slight decreases in out-of-sample accuracy and TDR for both methods due to the noise. The relative performance of the methods were almost exactly the same as on the noiseless data. For this reason, we have not included these results in full detail, but confirm that the trends we have shown indeed remain in the presence of noise.

Summary

We conclude by summarizing the collective findings of these tests with synthetic data. In every case, the trees generated by OCT were a closer match to the ground truth trees than those of CART. The out-of-sample accuracies of each method decreased with problem difficulty (decreasing training set size and increasing dimension) as we might expect, but OCT consistently gave accuracies 1–2% higher than CART. There was a similar trend in the true discovery rate, which represented the proportion of splits in the true tree that were correctly recovered, with OCT improving upon CART by about 2–5% across all the experiments. The biggest difference between the methods was in the false discovery rate, which represented the proportion of splits in the generated tree that were simply noise and unrelated. The CART trees had false discovery rates of 35–40% in all experiments, whereas the OCT trees were typically less than half this amount at 15–20%. Importantly, these high rates in the CART did not improve even as the training set became very large, indicating that CART is unable to filter out this noise from its trees, and CART trees inherently have a large fraction of irrelevant splits. In contrast, the false discovery rate of OCT decreases as the amount of data increases, so it is able to correctly learn the truth in the data without introducing unnecessary splits into the tree. Finally, we note that the significant advantage of OCT over CART persisted even in the presence of large levels of noise in the training data. This provides strong evidence against the widely-held belief that optimal methods are more inclined to overfit the training set at the expense of out-of-sample accuracy.

2.6 Experiments with Real-World Datasets

In this section, we compare the performance of Optimal Classification Trees and CART on several publicly available classification datasets widely-used within the statistics and machine learning communities.

Experimental Setup

In order to provide a comprehensive benchmark of real-world performance, we used a collection of 60 datasets obtained from the UCI Machine Learning Repository [77]. These datasets are used regularly for reporting and comparing the performance of different classification methods. The datasets we use have sizes up to hundreds of thousands of points, which we demonstrate can be handled by our methods.

Each dataset was split into three parts: the training set (50%), the validation set (25%), and the testing set (25%). The training and validation sets were used to tune the values of the hyperparameters of each method. We then used these tuned values to train a tree on the combined training and validation sets, which we then evaluate against the testing set to determine the out-of-sample accuracy.

We used the same implementations for OCT and CART as in Section 2.5. We trained Optimal Classification Trees of maximum depths from 1–10 on all datasets with $R = 500$ random restarts, using Algorithm 2.7 to tune the complexity parameter α and the depth of the tree. The CART trees were obtained by fixing `minbucket` $= 1$ to match OCT, and tuning the complexity parameter using cost-complexity pruning in the usual way for CART. The resulting tree was then trimmed to the specified depth if required. This lets us evaluate the performance of all methods in the same depth-constrained scenario, which is often important for applications that require interpretability of the solutions, and also to examine how the performances change with increasing depth.

We also conducted experiments where we allowed the depth of the trees to be greater than 10, however this did not lead to deeper trees in nearly all cases because the validation process controls the depth of the trees and prevents overfitting with trees that are too deep. This means the results for trees of depth 10 are nearly identical to the results that would be obtained with no depth restriction. As such, we only report the results for trees with maximum depths from 1–10.

In order to minimize the effect of the particular splitting of the data into training, validation and testing sets, the entire process was conducted five times for each dataset, with a different splitting each time. The final out-of-sample accuracies were then obtained by averaging the results across these five runs.

Optimal Classification Trees vs. CART

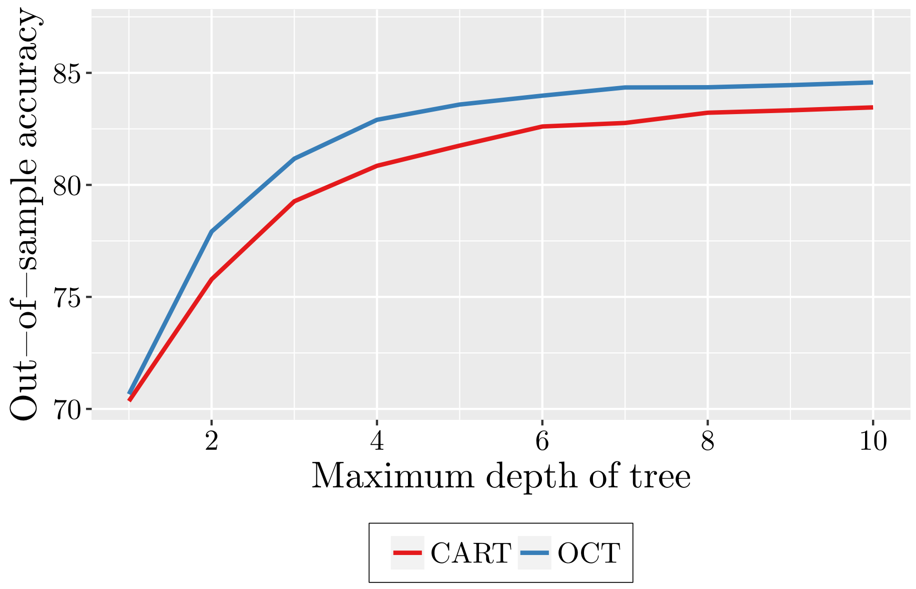
We now present an in-depth direct comparison of OCT and CART. Both methods seek to solve the same decision tree problem, so this comparison aims to show the effectiveness of solving the decision tree problem using modern optimization techniques to obtain more optimal solutions.

The entire set of mean out-of-sample accuracies on each dataset for both methods at depth 10 is provided in Table 2.1, which reports the size and dimension of the dataset, the number of class labels K, and the average out-of-sample accuracy of each

Table 2.1: Full results for CART and OCT at depth 10. The best method for each dataset is indicated in bold. Positive improvements are highlighted in blue, and negative in red.

Dataset				Mean out-of-sample accuracy		Mean improvement
Name	n	p	K	CART	OCT	
acute-inflammations-1	120	6	2	95.3	**100.0**	$+4.67 \pm 2.91$
acute-inflammations-2	120	6	2	**100.0**	**100.0**	0.00 ± 0.00
balance-scale	625	4	3	78.1	**79.4**	$+1.27 \pm 1.08$
banknote-authentication	1372	4	2	98.0	**98.9**	$+0.87 \pm 0.36$
blood-transfusion	748	4	2	77.2	**78.1**	$+0.86 \pm 0.80$
breast-cancer-diagnostic	569	30	2	90.9	**92.7**	$+1.82 \pm 0.28$
breast-cancer-prognostic	194	32	2	**75.5**	**75.5**	0.00 ± 0.00
breast-cancer	683	9	2	95.0	**95.1**	$+0.12 \pm 0.47$
car-evaluation	1728	15	4	91.3	**92.4**	$+1.16 \pm 1.18$
chess-king-rook-vs-king-pawn	3196	37	2	98.9	**99.3**	$+0.43 \pm 0.11$
climate-model-crashes	540	18	2	91.3	**92.3**	$+1.04 \pm 0.90$
congressional-voting-records	232	16	2	**98.6**	**98.6**	0.00 ± 0.55
connectionist-bench-sonar	208	60	2	69.2	**75.0**	$+5.77 \pm 2.51$
contraceptive-method-choice	1473	11	3	**53.2**	53.2	-0.05 ± 1.75
credit-approval	653	37	2	**87.2**	86.7	-0.49 ± 0.83
cylinder-bands	277	484	2	66.1	**67.5**	$+1.45 \pm 2.59$
dermatology	358	34	6	95.1	**95.3**	$+0.22 \pm 0.90$
echocardiogram	61	6	2	72.0	**73.3**	$+1.33 \pm 1.33$
fertility	100	12	2	**87.2**	86.4	-0.80 ± 1.96
haberman-survival	306	3	2	**73.5**	73.2	-0.26 ± 0.26
hayes-roth	132	4	3	75.8	**78.2**	$+2.42 \pm 4.64$
heart-disease-cleveland	297	18	5	**57.6**	55.5	-2.13 ± 1.00
hepatitis	80	19	2	**84.0**	82.0	-2.00 ± 3.00
hill-valley-noise	606	100	2	55.8	**56.6**	$+0.79 \pm 1.38$
hill-valley	606	100	2	49.8	**52.6**	$+2.78 \pm 0.68$
image-segmentation	210	19	7	77.7	**86.8**	$+9.06 \pm 0.71$
indian-liver-patient	579	10	2	**71.6**	71.2	-0.41 ± 0.60
ionosphere	351	34	2	88.3	**91.3**	$+2.99 \pm 3.12$
iris	150	4	3	93.5	**94.6**	$+1.08 \pm 1.08$
magic-gamma-telescope	19020	10	2	**84.9**	84.7	-0.24 ± 0.26
mammographic-mass	830	10	2	**81.7**	80.5	-1.26 ± 0.71
monks-problems-1	124	11	2	74.8	**87.7**	$+12.90 \pm 6.77$
monks-problems-2	169	11	2	59.5	**60.0**	$+0.47 \pm 0.47$
monks-problems-3	122	11	2	**94.2**	92.9	-1.29 ± 1.29
mushroom	5644	76	2	100.0	**100.0**	$+0.04 \pm 0.04$
optical-recognition	3823	64	10	**88.6**	88.0	-0.61 ± 0.15
ozone-level-detection-eight	1847	72	2	**93.1**	93.0	-0.09 ± 0.09
ozone-level-detection-one	1848	72	2	96.5	**96.6**	$+0.13 \pm 0.13$
parkinsons	195	21	2	**87.8**	86.9	-0.82 ± 4.21
pen-based-recognition	7494	16	10	**96.1**	95.5	-0.61 ± 0.19
pima-indians-diabetes	768	8	2	72.3	**73.0**	$+0.73 \pm 0.98$
planning-relax	182	12	2	**71.1**	**71.1**	0.00 ± 0.00
qsar-biodegradation	1055	41	2	82.4	**83.5**	$+1.14 \pm 1.63$
seeds	210	7	3	**89.4**	88.3	-1.13 ± 1.28
seismic-bumps	2584	20	2	92.9	**93.2**	$+0.31 \pm 0.48$
skin-segmentation	245057	3	2	99.7	**99.9**	$+0.17 \pm 0.02$
soybean-small	47	37	4	**100.0**	**100.0**	0.00 ± 0.00
spambase	4601	57	2	91.5	**93.2**	$+1.70 \pm 0.34$
spect-heart	80	22	2	61.0	**62.0**	$+1.00 \pm 4.00$
spectf-heart	80	44	2	72.0	**76.0**	$+4.00 \pm 6.20$
statlog-german-credit	1000	48	2	**72.0**	71.0	-1.04 ± 1.15
statlog-landsat	4435	36	6	85.3	**85.7**	$+0.45 \pm 0.57$
teaching-assistant	151	52	3	56.8	**63.2**	$+6.49 \pm 2.78$
thoracic-surgery	470	24	2	**84.8**	**84.8**	0.00 ± 0.00
thyroid-disease-ann	3772	21	3	**99.7**	99.7	-0.02 ± 0.04
thyroid-disease-new	215	5	3	92.8	**94.0**	$+1.13 \pm 0.75$
tic-tac-toe-endgame	958	18	2	**94.1**	94.0	-0.08 ± 0.89
wall-following-robot-2	5456	2	4	**100.0**	**100.0**	0.00 ± 0.00
wall-following-robot-24	5456	4	4	**100.0**	**100.0**	0.00 ± 0.00
wine	178	13	3	84.9	**94.2**	$+9.33 \pm 2.47$

Figure 2-17: Mean out-of-sample accuracy for each method across all 60 datasets.



method along with the mean accuracy improvement for OCT over CART and the corresponding standard error. As we noted earlier, the results with no depth restriction were nearly identical to the results for depth 10, so these results for depth 10 should additionally be seen as the results for the case where the depth is unconstrained.

Figure 2-17 shows the mean out-of-sample accuracy for each method across all 60 datasets as a function of the depth of the tree. These accuracies are also shown in Table 2.2, along with the mean difference between the methods and the associated p-value indicating the statistical significance of the difference.

We see that OCT is stronger than CART at all depths, with an improvement over CART of about 2% at lower depths, shrinking to about 1% at higher depths, and this difference is statistically significant at all depths. We believe that the difference is larger for the shallower trees because the impact of making a bad choice high in the tree is higher for shallow trees than for deeper trees, where there is more scope for the tree to recover from a bad initial split with the subsequent splits. Note that even at depth 1, a significant difference is present between the methods, which is simply the effect of using the misclassification score to select the best split as opposed to an impurity measure. Finally, we see that CART at depth 10 (or no depth restriction) performs about the same as OCT at depth 4, showing that OCT can learn trees of

Table 2.2: Mean out-of-sample accuracy (%) results for CART and OCT across all 60 datasets.

Maximum depth	Mean out-of-sample accuracy (%)		Mean improvement	p-value
	CART	OCT		
1	70.35	70.66	$+0.31 \pm 0.16$	0.0481
2	75.78	77.92	$+2.13 \pm 0.32$	$\sim 10^{-10}$
3	79.27	81.17	$+1.90 \pm 0.38$	$\sim 10^{-6}$
4	80.85	82.91	$+2.05 \pm 0.37$	$\sim 10^{-7}$
5	81.75	83.59	$+1.83 \pm 0.38$	$\sim 10^{-6}$
6	82.61	83.98	$+1.38 \pm 0.31$	$\sim 10^{-5}$
7	82.76	84.35	$+1.58 \pm 0.30$	$\sim 10^{-7}$
8	83.22	84.36	$+1.14 \pm 0.28$	$\sim 10^{-4}$
9	83.33	84.45	$+1.12 \pm 0.27$	$\sim 10^{-4}$
10	83.46	84.57	$+1.11 \pm 0.27$	$\sim 10^{-4}$

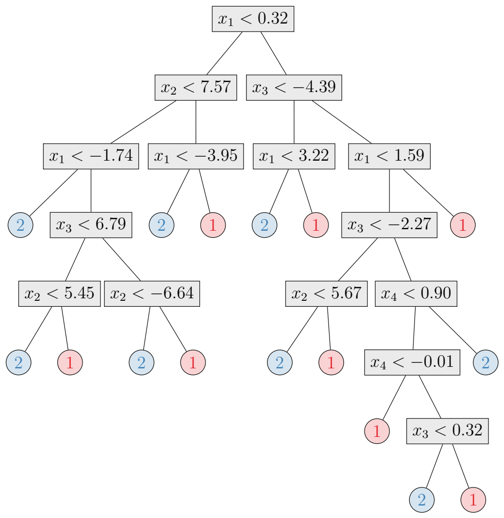
the same accuracy as CART with much fewer splits. This indicates that OCT is able to better discover the truth in the data and give trees with truly meaningful splits.

Figure 2-18 provides an example that emphasizes the additional power and interpretability of OCT over CART. The trees shown in this figure were generated as part of these computational experiments for the “Banknote Authentication” dataset with one of the five random seeds tested. From our results for this seed, we identified the CART tree with the best out-of-sample accuracy, which was 98.3%. This tree is shown in Figure 2-18a, and is depth 7 with 15 splits. The best out-of-sample accuracy for OCT with this seed was 99.1%, halving the error compared to the best CART tree. This tree is shown in Figure 2-18b, and is depth 6 with 12 splits, so not only provides a significant improvement in out-of-sample accuracy but also is slightly more interpretable since it is smaller. To provide a direct comparison of interpretability, we identified the depth where the OCT out-of-sample accuracy was closest to that of CART. This tree is shown in Figure 2-18c, and has the same out-of-sample accuracy as CART of 98.3% with depth 4 and 7 splits. OCT therefore achieves the same accuracy as CART with a tree that is approximately half the size, demonstrating the significant improvements in interpretability that are enabled by OCT.

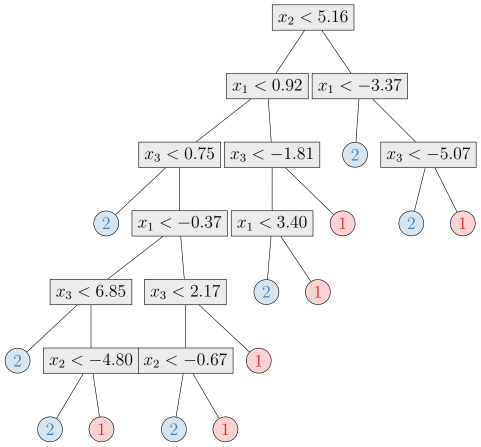
Table 2.3 shows the number of times CART and OCT were the strongest method

Figure 2-18: A comparison of CART and OCT trees on the "Banknote Authentication" dataset.

(a) CART (out-of-sample accuracy 98.3%; depth 7 with 15 splits)



(b) OCT (out-of-sample accuracy 99.1%; depth 6 with 12 splits)



(c) OCT (out-of-sample accuracy 98.3%; depth 4 with 7 splits)

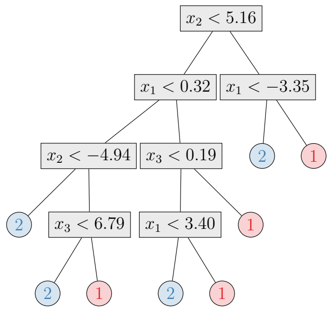


Table 2.3: Number of datasets where CART and OCT were strongest across all 60 datasets.

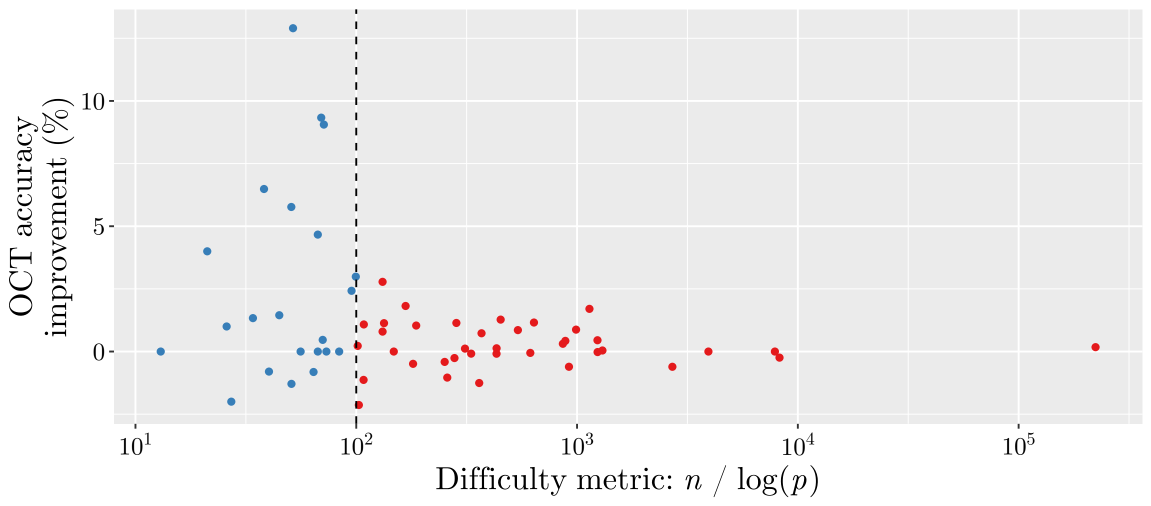
Maximum depth	CART	OCT	Ties
1	9	25	26
2	10	33	17
3	15	34	11
4	12	38	10
5	15	38	7
6	15	37	8
7	15	38	7
8	15	37	8
9	13	38	9
10	18	34	8

for a dataset across the sample of 60 datasets, broken down according to the maximum depth of the trees. We see that OCT performs strongest on significantly more datasets than CART. At depths 1 and 2, OCT wins about three times more often than CART, although there are a larger number of ties at these depths. At most other depths, OCT wins around 2.5 times more often than CART. The weakest performance for OCT relative to CART is at depth 10 (or with no depth restriction), yet it still performs strongest in about twice as many datasets as CART. These results show that OCT is able to outperform CART in a significant majority of datasets, in addition to having a higher mean out-of-sample accuracy across the sample of datasets.

We have established that OCT is more likely to outperform CART, and has a small yet significant gain in out-of-sample accuracy across the collection of datasets. Next, we consider the relative performance of the methods according to characteristics of the dataset in order to identify the types of problems where each method is more likely to outperform the other.

In the synthetic experiments of Section 2.5, we found that the improvement of OCT over CART increased as the problem difficulty, i.e., as the number of points decreased or as the number of features increased. We cannot easily investigate these effects individually for the real-world datasets since they are so varied in both n and p across the sample of datasets. Instead, we can construct a single measure for

Figure 2-19: Improvement in out-of-sample accuracy for OCT over CART for each of the 60 datasets, according to ratio of n and $\log(p)$.



problem difficulty, and measure the change in accuracy improvement as a function of this measure. The metric we have derived is $n/\log(p)$, which decreases as either n decreases or p increases, so lower values of this measure indicate more difficult problems, in line with the findings of the synthetic experiments. We found empirically that including the log in the denominator achieves the best balance between the scaling with n and p.

Figure 2-19 presents the accuracy improvement of OCT against CART as a function of this difficulty metric. We also divide the datasets into two groups: those with difficulty metric lower than 100, and those greater than 100. We can see that OCT gives the largest improvements in accuracy over CART for datasets below this threshold. The mean accuracy improvement for datasets below the threshold is 2.59%, while for those above the threshold the mean improvement is 0.26%. This reinforces the results from the synthetic experiments that the degree of improvement of OCT over CART is correlated with the problem difficulty; for difficult problems with few data or many features, the improvement of OCT tends to be larger.

We conclude this section by summarizing the key results from experiments on real-world datasets. Across a wide range of datasets with varying characteristics, we have comprehensive evidence that OCT delivers a significant improvement in out-of-sample accuracy over CART. In particular, when using trees of maximum depth 4, OCT can achieve the same accuracy as CART with no depth restriction. This demonstrates that OCT can achieve the same level of performance as CART with much less complexity in the tree, making the trees superior for interpretation. Finally, we found further evidence to reinforce the findings of the synthetic experiments in Section 2.5 that the improvement of OCT over CART increases as the problem difficulty increases, meaning much better at constructing trees for problems with fewer data or more features in the data.

2.7 Conclusions

In this chapter, we have revisited the classical problem of decision tree creation under a modern optimization lens. We framed the problem that the CART algorithm solves as a traditional optimization problem, and presented a novel MIO formulation for creating optimal decision trees that overcomes the inherent limitations of CART and other methods that are based on top-down, greedy induction.

We found empirically that the MIO formulation was too large to solve in practical times, and in a time-limited scenario delivered marginal improvements in solution quality but was not able to say anything about the optimality of these solutions after multiple hours. This motivated a new perspective on the problem to reduce the dimensionality, and led to a local search heuristic based around re-optimizing existing tree solutions one node at a time. This local search approach delivers solutions that significantly outperform both CART and the MIO-based approach, and runs in times within 1–2 orders of magnitude of CART.

We presented batch cost-complexity pruning, a new method for tuning the complexity parameter that gives much higher precision in choosing the optimal hyperparameter value compared to the traditional cost-complexity pruning. This new procedure combines the pruning results of multiple high-quality trees, which smoothens the resulting pruning curve and allows more accuracy in identifying the true minimizing value. We incorporated this method in a complete procedure for tuning the OCT hyperparameters that gives very strong out-of-sample performance without taking too long to tune.

Experiments with synthetic data provide strong evidence that OCT can better recover the true generating decision tree in the data, contrary to the popular belief that optimal methods will just overfit the training data at the expense of generalization ability. In particular, we find that around 35–40% of the splits in trees generated by CART bear no resemblance to the tree that generated the data, whereas this number is only around 15–20% for OCT, indicating the latter are much more reliable for interpretation.

We also conducted comprehensive computational experiments with a sample of 60 real-world datasets, and found that OCT significantly outperformed CART across this sample, with an average improvement in out-of-sample accuracy of 1–2% depending on the depth of the tree used. We also found that the improvement of OCT over CART increased as the problem difficulty increases, namely as the number of points decreases or the number of features increases. This indicates that OCT is more capable of identifying the true structure in the data in scenarios where there is much more noise than signal, a common occurrence in practical applications.

These results provide comprehensive evidence that the optimal decision tree problem is tractable for practical applications and leads to significant improvements over the current state-of-the-art decision tree learners.

Chapter 3

Optimal Classification Trees with Hyperplane Splits

Classical state-of-the-art methods for constructing decision trees focus exclusively on trees with splits that are parallel to the axes, regardless of whether they are single-tree methods like CART and C4.5, or ensemble methods like Random Forests and Gradient Boosting. In many cases, removing this restriction that the splits be parallel to the axes could lead to significant improvements in the quality of the tree. However, despite work in this area, to date no method has been developed to find trees with such hyperplane splits that is both tractable on real problem sizes and leads to significant improvements over the axes-parallel case.

The key problem when trying to generate trees with hyperplane splits in the same way as axis-parallel splits is the large number of possible hyperplane splits. The axis-parallel algorithms find the optimal split at a node by conducting an exhaustive search of all $n \cdot p$ possible splits. When we remove the axis-parallel restriction, this number of possible splits increases to $2^p \cdot \binom{n}{p}$, because every subset of size p from the n points defines a p-dimensional hyperplane which can then be rotated slightly to divide the subset of p points in all 2^p ways. This large increase in the number of possible splits makes an exhaustive search prohibitively expensive; the problem of finding the hyperplane split with the lowest misclassification error is NP-hard [59].

There have been many proposals that yield approximate solutions for the best hyperplane split. The first method for trees with hyperplane splits was CART with linear combinations (CART-LC) [25], and works by perturbing the coefficients in the hyperplane sequentially until a local optimum is attained. The main limitation to this method is that the search procedure is deterministic and always converges to the same optimum for a given set of points, which may be far from the global optimum. Another method using a perturbation-based approach is SADT (Simulated Annealing for Decision Trees) [58] which uses simulated annealing to perturb an existing hyperplane towards optimality. These perturbation approaches were combined to give OC1 [91], which first uses the deterministic procedure of CART-LC to find a local optimum, followed by the randomness of SADT to escape this local optimum. This is coupled with searching from multiple randomized starting points to increase the chances of finding a high-quality local optimum. Most other approaches for trees with hyperplane splits apply other methods recursively to generate the splits in the tree, such as logistic regression [116], support vector machines [9], linear discriminant analysis [82, 83], and Householder transformations [120].

Most of these approaches do not have easily accessible implementations that can be used on practically-sized datasets and as such, the use of decision trees with hyperplanes in the statistics/machine learning community has been limited. We also note that these approaches share the same major flaw as univariate decision trees, in that the splits are formed one-by-one using top-down induction, and so the choice of split cannot be guided by the possible influence of future splits.

In this chapter, we consider generating optimal classification trees with hyperplanes. Specifically, we extend our approaches from Chapter 2 to generate trees with hyperplane splits, and show that these trees significantly improve upon state-of-the-art methods.

3.1 OCT with Hyperplanes via MIO

In Section 2.2, we developed an MIO formulation for finding the optimal classification tree. This formulation only considered decision trees that use a single variable in their splits at each node. In this section, we show that it is simple to extend our MIO formulation for axis-parallel trees to yield a problem for determining the optimal decision tree with hyperplanes. When viewed from an MIO perspective, the axis-parallel and hyperplane problems are very similar, and modeling the hyperplane problem only requires minor modifications to the formulation for the axis-parallel problem. This shows the flexibility and power of modeling the problem using MIO.

In a decision tree with hyperplanes, we are no longer restricted to choosing a single variable upon which to split, and instead can choose a hyperplane split at each node. The variables $\mathbf{a}_t$ will be used to model the split at each node as before, except we relax (2.4) and instead choose $\mathbf{a}_t \in [-1, 1]^p$ at each branch node t. We must modify (2.2) to account for the the possibility these elements are negative by dealing with the absolute values instead:

$$\sum_{j=1}^{p} |a_{jt}| \leq d_t, \quad \forall t \in \mathcal{T}_B,$$

which can be linearized using auxiliary variables to track the value of $|a_{jt}|$:

$$\begin{aligned} &\sum_{j=1}^{p} \hat{a}_{jt} \leq d_t, && \forall t \in \mathcal{T}_B, \\ &\hat{a}_{jt} \geq a_{jt}, && \forall t \in \mathcal{T}_B,\ j \in [p], \\ &\hat{a}_{jt} \geq -a_{jt}, && \forall t \in \mathcal{T}_B. \end{aligned}$$

As before, these constraints force the split to be all zeros if $d_t = 0$ and no split is applied, otherwise imposing no restriction on $\mathbf{a}_t$.

We now have that $\mathbf{a}_t^T \mathbf{x}_i \in [-1, 1]$, so we replace (2.3) with:

$$-d_t \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B.$$

Now we consider the split constraints (2.9) and (2.14). Previously we had that the range of $(\mathbf{a}_t^T \mathbf{x}_i - b_t)$ was $[-1, 1]$, whereas it is now $[-2, 2]$. This means we need $M = 2$

to ensure that the constraint is trivially satisfied when $z_{it} = 0$. The constraints therefore become:

$$\mathbf{a}_m^T \mathbf{x}_i < b_m + 2(1 - z_{it}), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t), \tag{3.1}$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - 2(1 - z_{it}), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t), \tag{3.2}$$

Finally, as before we need to convert the strict inequality in (3.1) to a non-strict version. We do this by introducing a sufficiently small constant μ:

$$\mathbf{a}_m^T \mathbf{x}_i + \mu \leq b_m + (2 + \mu)(1 - z_{it}), \quad \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t),$$

Note that we need to include μ in the rightmost term to ensure the constraint is always satisfied when $z_{it} = 0$. Unlike in the axis-parallel case, we do not choose a value for μ in an intelligent manner, and instead need to choose a small constant. This is because a hyperplane that separates two points can be rotated to come arbitrarily close to each point, and it is not always feasible to find the minimum separation distance like for axis-parallel splits as this would require enumeration of all $2^p \cdot \binom{n}{p}$ hyperplane splits. We must take care when choosing the value of μ. A value that is too small can lead to numerical issues in the MIO solver, while one that is too large reduces the size of the feasible region, potentially reducing the quality of the optimal solution. We take $\mu = 0.005$ as a compromise between these extremes.

In the axis-parallel problem, we controlled the complexity of the tree by penalizing the number of splits. In the hyperplane regime, a single split may use multiple variables, and it seems natural to treat splits with greater numbers of variables as more complex than those with fewer. To achieve this, we can instead penalize the total number of variables used in the splits of the tree, which we note in the axis-parallel case is exactly the number of splits in the tree. To achieve this, we introduce binary variables s_{jt} to track if the jth feature is used in the tth split:

$$-s_{jt} \leq a_{jt} \leq s_{jt}, \quad \forall t \in \mathcal{T}_B,\ j \in [p].$$

We must also make sure that the values of s_{jt} and d_t are compatible. The following constraints ensure that $d_t = 1$ if and only if any variable is used in the split:

$$\begin{aligned}
& s_{jt} \leq d_t, && \forall t \in \mathcal{T}_B,\ j \in [p], \\
& \sum_{j=1}^{p} s_{jt} \geq d_t, && \forall t \in \mathcal{T}_B.
\end{aligned}$$

Finally, we modify the definition of complexity C to penalize the number of variables used across the splits in the tree rather than simply the number of splits:

$$C = \sum_{t \in \mathcal{T}_B} \sum_{j=1}^{p} s_{jt}$$

Combining all of these changes yields the complete *Optimal Classification Trees with Hyperplanes* (OCT-H) model:

$$\begin{aligned}
\min \quad & \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \ \alpha \cdot C && && (3.3) \\
\text{s.t.} \quad & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), && \forall k \in [K],\ t \in \mathcal{T}_L, \\
& L_t \leq N_t - N_{kt} + n c_{kt}, && \forall k \in [K],\ t \in \mathcal{T}_L, \\
& L_t \geq 0, && \forall t \in \mathcal{T}_L, \\
& N_{kt} = \sum_{i:\ y_i = k} z_{it}, && \forall k \in [K],\ t \in \mathcal{T}_L, \\
& N_t = \sum_{i=1}^{n} z_{it}, && \forall t \in \mathcal{T}_L, \\
& \sum_{k=1}^{K} c_{kt} = l_t, && \forall t \in \mathcal{T}_L, \\
& C = \sum_{t \in \mathcal{T}_B} \sum_{j=1}^{p} s_{jt}, \\
& \mathbf{a}_m^T \mathbf{x}_i + \mu \leq b_m + (2 + \mu)\left(1 - z_{it}\right), && \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t), \\
& \mathbf{a}_m^T \mathbf{x}_i \geq b_m - 2\left(1 - z_{it}\right), && \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t), \\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, && \forall i \in [n],
\end{aligned}$$

$$
\begin{aligned}
& z_{it} \leq l_t, && \forall t \in \mathcal{T}_L, \\
& \sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, && \forall t \in \mathcal{T}_L, \\
& \sum_{j=1}^{p} \hat{a}_{jt} \leq d_t, && \forall t \in \mathcal{T}_B, \\
& \hat{a}_{jt} \geq a_{jt}, && \forall t \in \mathcal{T}_B,\ j \in [p], \\
& \hat{a}_{jt} \geq -a_{jt}, && \forall t \in \mathcal{T}_B,\ j \in [p], \\
& -s_{jt} \leq a_{jt} \leq s_{jt}, && \forall t \in \mathcal{T}_B,\ j \in [p], \\
& s_{jt} \leq d_t, && \forall t \in \mathcal{T}_B,\ j \in [p], \\
& \sum_{j=1}^{p} s_{jt} \geq d_t, && \forall t \in \mathcal{T}_B, \\
& -d_t \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B, \\
& d_t \leq d_{p(t)}, && \forall t \in \mathcal{T}_B \setminus \{1\}, \\
& z_{it}, l_t, c_{kt} \in \{0,1\}, && \forall i \in [n],\ k \in [K],\ t \in \mathcal{T}_L, \\
& d_t, s_{jt} \in \{0,1\}, && \forall j \in [p],\ t \in \mathcal{T}_B.
\end{aligned}
$$

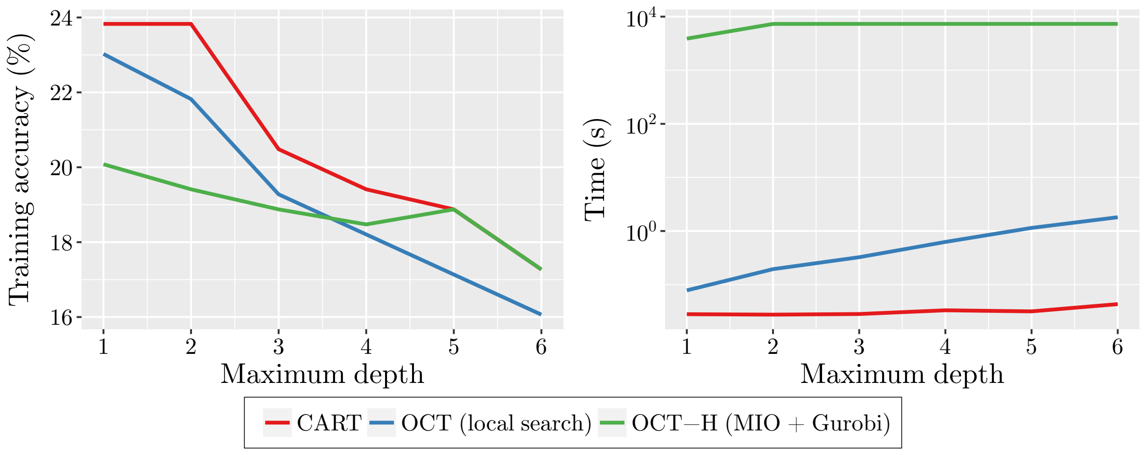
Note that from this formulation we can easily obtain the OCT problem as a special case by restoring the binary constraints on $\mathbf{a}_t$. The close relationship between the axis-parallel and hyperplane problems reinforces the notion that the MIO formulation is a natural way of viewing the decision tree problem.

3.2 OCT with Hyperplanes via Local Search

In this section, we extend the local search heuristic presented in Section 2.3 to generate trees with hyperplane splits, thereby avoiding the scaling issues of MIO-based approaches and allowing us to generate Optimal Classification Trees with Hyperplanes in a tractable and effective manner.

The MIO formulation for the OCT-H problem exhibits the same scaling issues as in the parallel split case. The number of variables in the formulation grows incredibly

Figure 3-1: Training accuracy (%) and running time for each method on "Blood Transfusion Service Center" dataset.



fast with the depth of the tree and number of training points, quickly causing the MIO problem to become intractable. Figure 3-1 shows the performance of this MIO formulation for OCT-H on the "Blood Transfusion Service Center" dataset from the UCI Machine Learning Repository [77], along with CART and OCT using local search. This small dataset has $n = 748$ points with $p = 4$ features and $K = 2$ classes. We can see at low depths that the OCT-H trees with hyperplane splits are significantly stronger than the other trees which use parallel splits, which we would expect as we know hyperplane splits are more powerful. However, as the depth increases, the accuracy does not improve as fast as for the parallel tree methods, and at depths 5 and 6 does not improve upon the CART warm start at all. This is clear indication that the problem is too large and difficult for the solver to find improved solutions, as we expect hyperplane trees to be at least as powerful as the corresponding parallel trees at each depth. We also can see that the MIO-based method is again many orders of magnitude slower than the others, which limits its potential for practical application.

To resolve the performance issues of the MIO-based method, we will use a local-search heuristic similar to the one presented in Section 2.3 but with modifications to allow generation trees with hyperplane splits. To do this, we simply augment the procedure OPTIMIZENODEPARALLEL from Algorithm 2.2 to search for the best hyperplane split in addition to the existing steps.

We use a perturbation approach to find optimize the coefficients in the hyperplane split, similar to that employed by CART-LC [25] and OC1 [91]. Underpinning this approach is the observation that it is possible to optimize the value of a single coefficient at a time using a linear search in $O(n)$ time. To see this, consider the hyperplane split defined by $\hat{\mathbf{a}}$ and $\hat{b}$. A point $\mathbf{x}_i$ will follow the lower branch of this split if

$$\hat{V}_i \triangleq \sum_{j=1}^{p} \hat{a}_j x_{ij} - \hat{b} < 0, \tag{3.4}$$

otherwise the upper branch will be taken.

Now consider changing the value of a single coefficient in the kth feature from $\hat{a}_k$ to a_k. The point $\mathbf{x}_i$ will now follow the lower branch of this split if

$$\hat{V}_i + (a_k - \hat{a}_k)x_{ik} < 0. \tag{3.5}$$

If $x_{ik} > 0$, the point i follows the lower split if and only if

$$a_k < \frac{\hat{a}_k x_{ik} - \hat{V}_i}{x_{ik}}, \tag{3.6}$$

and otherwise we have $x_{ik} = 0$ (since the data are normalized to $[0, 1]$) and so this test reduces to $\hat{V}_i < 0$ for point i to follow the lower split. Combining these, we have that point i follows the lower split if and only if $a_k < U_{ik}$, where

$$U_{ik} = \begin{cases} \dfrac{\hat{a}_k x_{ik} - \hat{V}_i}{x_{ik}}, & x_{ik} > 0, \\ -\infty, & x_{ik} = 0, V_i \geq 0, \\ +\infty, & x_{ik} = 0, V_i < 0. \end{cases} \tag{3.7}$$

We calculate these thresholds U_{ik} for every training point at the node under consideration, giving us the critical values at which points will change from one side of the hyperplane to the other. We sort these threshold values and then efficiently optimize the value of a_k over $\mathbb{R}$ by scanning the critical values in order and moving points from the lower branch to the upper branch one-by-one, calculating the resulting error at each step. We return the value of a_k that gave the lowest error, making this coefficient locally optimal. This search is nearly identical to the searches performed by both the standard CART algorithm and our procedure BestParallelSplit from Algorithm 2.3, except with the use of the values U_{ik} in place of x_{ik}.

We also consider changing the hyperplane split by removing features from split, which reduces the complexity so may improve the overall loss. To do this, we calculate the values of each point corresponding to the split after the kth coefficient has been deleted:

$$W_{ik} \triangleq \hat{V}_i + \hat{b} - \hat{a}_k x_{ik}, \tag{3.8}$$

and therefore point i follows the lower branch of the split with the kth coefficient deleted if $W_{ik} < b$. We can then sort these critical values and scan them in the same way as above to find the optimal value for b that minimizes the loss function. This gives us the optimal threshold for the hyperplane after the kth coefficient is deleted, which we accept if this transformation improves the loss.

We repeat this search to optimize the coefficients by perturbation and deletion in turn until no improvement is possible for any coefficient, giving us a hyperplane split that is a local minimum for the error at the node. The full details of this search procedure are given in Algorithm 3.1.

We find empirically that there are typically many such local minima at a node in the tree, and so which minimum is reached depends on the starting solution. We address this by considering perturbations from many starting hyperplane splits, similar to how the local search algorithm considers many starting trees. The first solution we consider is always the best parallel split at that node, ensuring that our final solution is at least as good as this best parallel split. We then also use H random hyperplane splits as starting solutions, where H is a parameter of the problem. Increasing the number of hyperplane restarts will generally improve the quality of the local minimum found at the node, but will take longer to run. We have found that setting H to 5–10 gives the best tradeoff between accuracy and runtime.

Algorithm 3.1 BestHyperplaneSplit

```
Input: Starting subtree $\mathbb{T}$; training data $\mathbf{X}$, $\mathbf{y}$
Output: Subtree $\mathbb{T}$ with high-quality hyperplane split at root; error of subtree $\mathbb{T}$
 1: repeat
 2:     $\text{error}_{\text{prev}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$
 3:     for $k = \text{shuffle}(1, \ldots, p)$ do                              ▷ loop over all dimensions
 4:         values $\leftarrow \{U_{ik} : i = 1, \ldots, n\}$                    ▷ critical values for perturbing
 5:         sort values in ascending order
 6:         for $i = 1, \ldots, n-1$ do                                        ▷ loop over all split placements
 7:             $c \leftarrow \frac{1}{2}(\text{values}_i + \text{values}_{i+1})$
 8:             replace $a_k$ in split at root of $\mathbb{T}$ with $c$
 9:             if $\text{minleafsize}(\mathbb{T}) \geq N_{\text{min}}$ then                         ▷ check feasibility
10:                 error $\leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$
11:                 if error $< \text{error}_{\text{best}}$ then                                ▷ save split if better
12:                     $\text{error}_{\text{best}} \leftarrow$ error
13:                     $\mathbb{T}_{\text{best}} \leftarrow \mathbb{T}$
14:         if $a_k \neq 0$ at root of $\mathbb{T}$ then                            ▷ can only delete if present
15:             values $\leftarrow \{W_{ik} : i = 1, \ldots, n\}$                   ▷ critical values for deleting
16:             sort values in ascending order
17:             for $i = 1, \ldots, n-1$ do                                    ▷ loop over all split placements
18:                 $b \leftarrow \frac{1}{2}(\text{values}_i + \text{values}_{i+1})$
19:                 replace $a_k$ in split at root of $\mathbb{T}$ with 0
20:                 change split threshold at root of $\mathbb{T}$ to $b$
21:                 if $\text{minleafsize}(\mathbb{T}) \geq N_{\text{min}}$ then                     ▷ check feasibility
22:                     error $\leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$
23:                     if error $< \text{error}_{\text{best}}$ then                            ▷ save split if better
24:                         $\text{error}_{\text{best}} \leftarrow$ error
25:                         $\mathbb{T}_{\text{best}} \leftarrow \mathbb{T}$
26: until $\text{error}_{\text{prev}} = \text{error}_{\text{best}}$                          ▷ no further improvement possible
27: return $\mathbb{T}, \text{error}_{\text{best}}$
```

This process is detailed in Algorithm 3.2, and the full local search procedure for trees with hyperplane splits is then simply the same as shown in Algorithm 2.1 with OptimizeNodeParallel replaced with OptimizeNodeHyperplane.

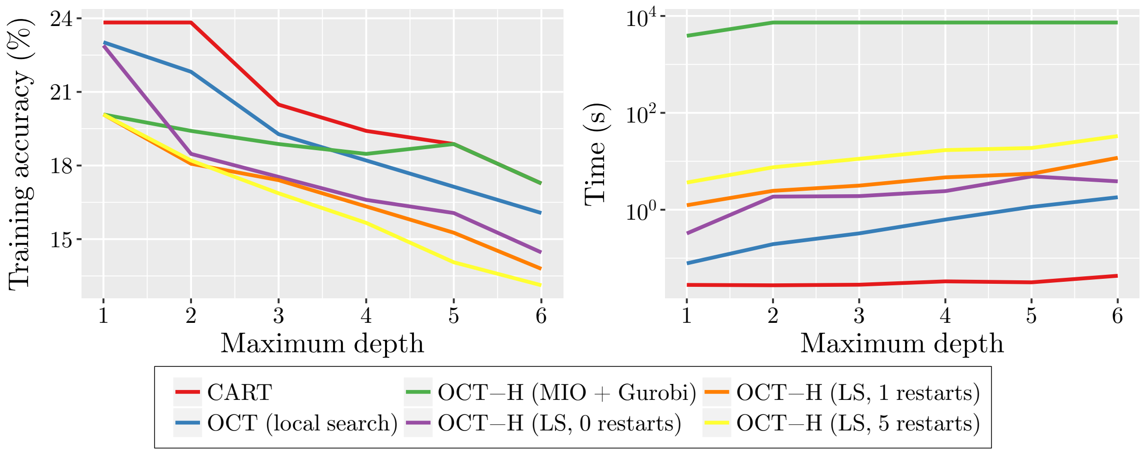
To examine the performance of this local search approach for hyperplane splits, we return the example using the “Blood Transfusion Service Center” dataset. Figure 3-2 shows the performance of the OCT-H local search method with $H = 0$, 1, and 5 random hyperplane restarts, alongside CART, OCT, and OCT-H with MIO. We see that the OCT-H methods using local search improve significantly upon the accuracy

Algorithm 3.2 OptimizeNodeHyper

```
Input: Subtree $\mathbb{T}$ to optimize; training data $\mathbf{X}$, $\mathbf{y}$
Output: Subtree $\mathbb{T}$ with best parallel or hyperplane split at root
 1: if $\mathbb{T}$ is a branch then
 2:     $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}} \leftarrow \text{children}(\mathbb{T})$
 3: else
 4:     $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}} \leftarrow$ new leaf nodes
 5: $\text{error}_{\text{best}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$          ▷ error of current root
 6:
 7: $\mathbb{T}_{\text{para}}, \text{error}_{\text{para}} \leftarrow \textsc{BestParallelSplit}(\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}, \mathbf{X}, \mathbf{y})$
 8: if $\text{error}_{\text{para}} < \text{error}_{\text{best}}$ then
 9:     $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{para}}, \text{error}_{\text{para}}$          ▷ replace with parallel split
10: $\text{error}_{\text{lower}} \leftarrow \text{loss}(\mathbb{T}_{\text{lower}}, \mathbf{X}, \mathbf{y})$
11: if $\text{error}_{\text{lower}} < \text{error}_{\text{best}}$ then
12:     $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{lower}}, \text{error}_{\text{lower}}$          ▷ replace with lower child
13: $\text{error}_{\text{upper}} \leftarrow \text{loss}(\mathbb{T}_{\text{upper}}, \mathbf{X}, \mathbf{y})$
14: if $\text{error}_{\text{upper}} < \text{error}_{\text{best}}$ then
15:     $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{upper}}, \text{error}_{\text{upper}}$          ▷ replace with upper child
16:
17: for $h = 1, \ldots, H + 1$ do
18:     if $h = 1$ then
19:         $\mathbb{T}_{\text{start}} \leftarrow \mathbb{T}_{\text{para}}$          ▷ use best parallel split as first start
20:     else
21:         generate random split $\mathbf{a}$, $b$
22:         $\mathbb{T}_{\text{start}} \leftarrow$ branch node $\mathbf{a}^\mathsf{T}\mathbf{x} < b$ with children $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}$
23:     $\mathbb{T}_{\text{hyper}}, \text{error}_{\text{hyper}} \leftarrow \textsc{BestHyperplaneSplit}(\mathbb{T}_{\text{start}}, \mathbf{X}, \mathbf{y})$
24:     if $\text{error}_{\text{hyper}} < \text{error}_{\text{hyper}}$ then
25:         $\mathbb{T}, \text{error}_{\text{best}} \leftarrow \mathbb{T}_{\text{hyper}}, \text{error}_{\text{hyper}}$          ▷ replace with hyperplane split
26: return $\mathbb{T}$
```

of the other three methods. The accuracy of OCT-H increases with the number of random hyperplane restarts, as we would expect since the splits we are finding are closer to optimality. Interestingly, OCT-H with no hyperplane restarts does not perform any better than OCT at depth 1 and is outperformed by the MIO-based OCT-H. However, once a single restart is added, the local search matches the accuracy of the MIO-based method. This would indicate that simply perturbing the best parallel split to find a hyperplane does not deliver solutions of the highest quality, and we can improve upon this easily and cheaply by adding random hyperplane restarts. In terms of runtime, the OCT-H methods take longer than OCT as we would expect,

Figure 3-2: Training accuracy (%) and running time for each method on "Blood Transfusion Service Center" dataset.



and the runtime increases as the number of hyperplane restarts increases. These increases are small relative to the runtime of the MIO-based OCT-H, and are still reasonable for practical applications, since OCT-H with 5 hyperplane restarts and 100 tree restarts takes only 30 seconds, and delivers significant improvements in return for this increased runtime. We also note as before that these runtimes are for a single-core scenario; in a multi-core setting the runtimes could be just a few seconds.

Complexity analysis of hyperplane local search

We finish the presentation of the local search algorithm for the OCT-H problem by deriving its time complexity. Recall in Section 2.3 we showed that the complexities of CART and OCT were $O(n^2p)$ and $O(n^3p)$, respectively.

First, we consider the BestHyperplaneSplit function in Algorithm 3.1, which is very similar to the BestParallelSplit in Algorithm 2.3. The biggest difference is that the U_{ik} and W_{ik} values have to be calculated using the current tree, and thus cannot be precomputed and presorted as in the parallel case. However, once we have these values sorted, the two inner loops are the same as for BestParallelSplit, and so have cost of $O(npK)$. We can calculate U_{ik} and W_{ik} together for all i and k in $O(np)$ time, and then sorting these in each feature takes $O(n \log n)$ time, for a total cost of $O(np) + p \cdot O(n \log n) = O(np \log n)$. This means the BestHyperplaneSplit function has a total cost of $O(np(K + \log n))$.

Next, we consider OptimizeNodeHyperplane from Algorithm 3.2, which is again nearly identical to its parallel counterpart OptimizeNodeParallel in Algorithm 2.2, which has a cost of $O(npK)$. The only difference is that we have added the final loop that generates hyperplane splits. This loop first generates a random hyperplane at cost $O(p)$, followed by running BestHyperplaneSplit at cost $O(np(K + \log n))$, so each iteration of this loop runs in $O(np(K + \log n))$ time. The loop runs for $H + 1$ iterations, where H is a constant and so does not affect the complexity. This means the total cost of the OptimizeNodeHyperplane is simply $O(np(K + \log n))$.

Finally, we run the LocalSearch function from Algorithm 2.1 with OptimizeNodeHyperplane in place of OptimizeNodeParallel. Using a similar approach as in the parallel case, we can compute point-to-leaf assignments and errors of all subtrees outside the inner loop, at a cost of $O(npT + KT) = O(npT)$, where the new factor of p relative to the parallel case comes from each split in the tree using up to p features for the hyperplane rather than one for parallel splits. The inner loop runs over T nodes as in the parallel case, giving a runtime of $O(npT) + T \cdot O(np(K + \log n)) = O(npKT + npT \log n)$ per local search iteration. Using a similar argument to before, we can bound the number of local search iterations by $O(n)$ if the complexity parameter α is zero, giving a final overall runtime for the local search of $O(n^2pT(K + \log n))$.

In the absolute worst case where T is $O(n)$, then the runtime of the local search for OCT-H would be $O(n^3p(K + \log n))$, which is very similar to the runtime of the OCT local search of $O(n^3pK)$, and so the addition of hyperplane splits has not significantly increased the worst-case cost.

Under our set of realistic assumptions that the number of nodes in the tree and the number of local search iterations are both $O(\log n)$, the OCT-H runtime would be $O(np \log^2 n(K + \log n))$, compared to $O(npK \log^2 n)$ for OCT and $O(npK \log n)$ for CART. As for the worst case, the only difference between OCT-H and OCT is replacing a factor of K with $(K + \log n)$.

3.3 The Accuracy-Interpretability Tradeoff

In this section, we discuss the tradeoff between the interpretability and accuracy of prediction models, first in the wider context of classification problems in general, and then in a comparison between decision trees with parallel and hyperplane splits.

The current state of the art in classification

For the past 30 years, CART has remained among the state-of-the-art methods for inducing decision trees. As a class of methods, decision tree learners are unparalleled in their interpretability; to quote Breiman, "On interpretability, trees rate an A+" [30]. They do not however achieve state-of-the-art accuracies that are competitive with the best methods for classification problems, although there are tree-based methods that do achieve such accuracies.

One such method is random forests [29], which works by generating a forest of CART trees, and then makes predictions by averaging the predictions of each tree in the forest. To increase variance among the trees in the forest, each tree is generated using a different bootstrap sample of the training points, and at each stage of the tree growing process, the feature selected for the split can only be chosen from a random sample of the features (typically of size $\sqrt{p}$). The trees in the forest can be trained independently, and so the whole training process is trivially parallelizable and can achieve runtimes comparable to CART.

Another tree-based method that gives state-of-the-art accuracy is gradient-boosted trees [49]. Like random forests, tree boosting also generates a collection of trees, except the trees are generated iteratively. The prediction of the tree ensemble is given by a weighted average of the predictions of the trees inside, and each new tree is trained to fit the residuals of the current ensemble before being added to the ensemble with a weight that minimizes the overall training error of the ensemble. In this way, the boosting process iteratively refines the predictions of the ensemble by fitting trees that focus on the points in the training set where the error is currently largest. Tree-boosting delivers accuracies that are among the highest for classification and regression problems, and there exist highly-optimized implementations such as XGBoost [35].

Both of these methods achieve state-of-the-art accuracies and are ensemble methods, meaning they averaging a large collection of trees, rather than generating a single decision tree like CART. The interpretability of these methods is therefore vastly diminished. This leads to the current dilemma that machine learning practioners face in applications: one can either use a method like CART that is interpretable but does not achieve a state-of-the-art accuracy, or one can use an ensemble method like random forests or tree boosting that delivers state-of-the-art accuracy but throws away interpretability.

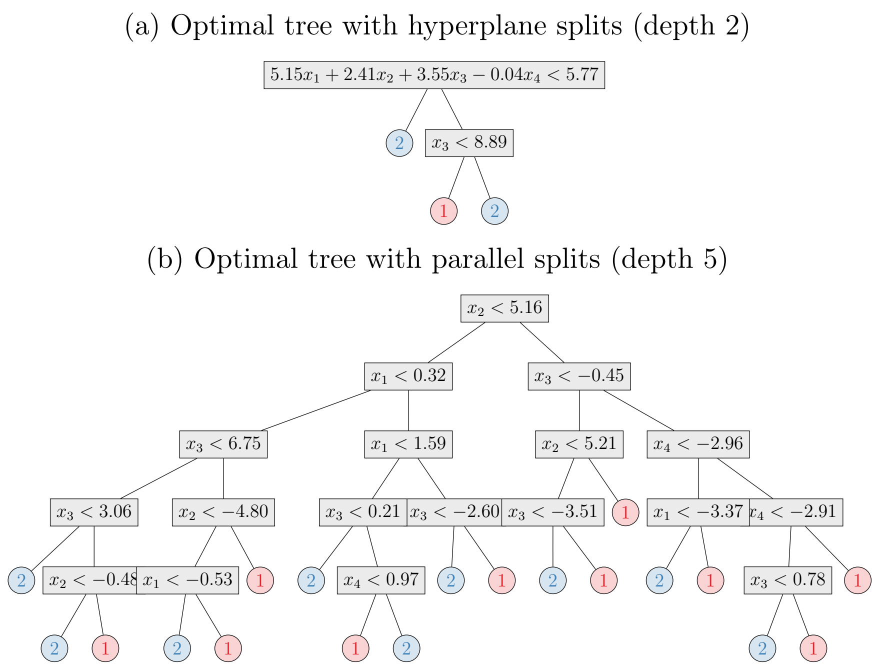
Our aspiration is that Optimal Trees can allow us to do away with this tradeoff entirely. Our goal is for the improved accuracy of Optimal Trees to reach state-of-the-art levels, thus making them competitive with random forests and boosting in terms of accuracy whilst maintaining the interpretability of a single decision tree.

Interpretability of Parallel and Hyperplane Trees

If we consider a single split in a decision tree, there is no question that a parallel split is more easily interpreted than a hyperplane split, particularly as the number of features used in the hyperplane split increases. As humans, we typically cannot easily make sense of the linear combination of features in a hyperplane split, instead we are better suited to interpreting just a single feature at a time. This means that when we consider a single split in a decision tree, the increased power of using a hyperplane split is balanced by the reduction in interpretability compared to a parallel split.

However, although hyperplane splits are less interpretable than parallel splits, it is not necessarily the case that trees with hyperplane splits are less interpretable than those with parallel splits. This is because the increased power of the hyperplane splits can enable the tree to achieve equivalent accuracies to parallel trees with fewer splits, and so the complexity of the tree is reduced. This means we need to consider the tradeoff of the reduction in split interpretability against the gain in interpretability through reduction of tree complexity.

Figure 3-3: Comparison of optimal trees with parallel and hyperplane splits on the “Banknote Authentication” dataset. The depth shown for each split type is the smallest depth that gave a tree with training error below 0.5%.



An example of such a scenario is shown in Figure 3-3, which shows two trees that were trained on the “Banknote Authentication” dataset from the UCI Machine Learning Repository [77], one with hyperplane splits and the other with parallel splits. We trained each of these trees by increasing the maximum depth until the training error fell below 0.5%. This gives us trees that have relatively few splits, yet still achieve near-perfect training error. We see that the tree with hyperplane splits is depth 2, and only has two splits, whereas the tree with parallel splits is depth 5 with 18 splits. Moreover, one of the splits in the hyperplane tree is actually a parallel split, so there is no reduction in interpretability. This clearly showcases the tradeoff between parallel and hyperplane trees; we can reduce the size and complexity of the tree by nearly 90% by using a single hyperplane split at the root, but in doing so we increase the complexity of this split reducing its interpretability.

The question of whether parallel or hyperplane trees are more appropriate for a given problem is very context-dependent. We expect hyperplane trees to have more power and hence higher accuracy, and this may come at the expense of some interpretability. However, as the example of Figure 3-3 shows, it is not always true that a tree with hyperplane splits is less interpretable than one with parallel splits, due to the reduction in tree size permitted by the increased power of hyperplane splits. We also note that the OCT-H model rewards sparsity in the hyperplane splits by penalizing the number of features used in each split. This means that the splits in OCT-H trees may not reduce interpretability too greatly, as the majority of these splits typically end up with only very few features.

3.4 Experiments with Synthetic Datasets

In this section, we investigate the performance of Optimal Classification Trees with Hyperplanes on a suite of experiments using synthetically-generated data. The goal of these experiments is to measure the performance of OCT-H relative to both OCT and CART to see the impact of allowing hyperplane splits, and also relative to random forests and boosting to evaluate OCT-H against methods with state-of-the-art accuracy.

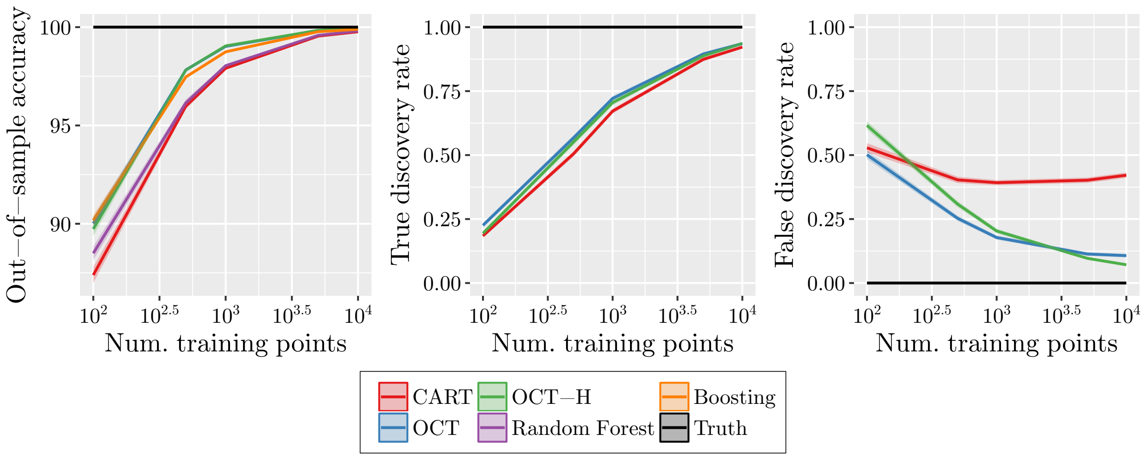
The experimental setup is the same as in Section 2.5. We randomly generate decision trees and then use these trees to generate training and testing datasets. Unless otherwise mentioned, we use the same default parameters as in Section 2.5 and use $H = 5$ random hyperplane restarts in the OCT-H method.

In each experiment, we compare the performance of CART, OCT and OCT-H, along with random forests (using the randomForest package in R [76]) and gradient-boosted trees (using the XGBoost library [35] with $\eta = 0.1$). The best depths for random forests and boosting were determined through validation, and we use the number of random restarts for the Optimal Tree methods as the number of trees for random forests and boosting.

Ground truth trees with parallel splits

The first set of experiments we conduct are the same as in Section 2.5, except with the addition of the three new methods. There are two main aims in repeating these tests

Figure 3-4: Synthetic experiments showing the effect of training set size.

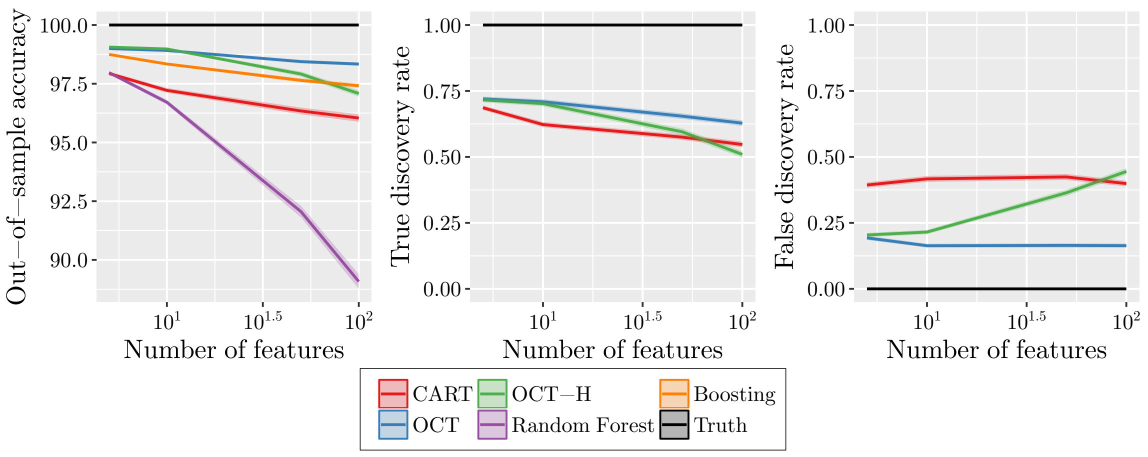


with the new methods. The first is to examine the performance of OCT-H on data that is generated from a parallel tree structure, and see whether it is able to train trees that give performance similar to OCT. It seems possible that the additional flexibility and power in the OCT-H model could lead it to overfit the training data with trees of high complexity, so it is important to see whether this is the case. The other goal is to benchmark random forests and boosting on these tree-structured datasets to give additional context to the performance results of OCT and OCT-H.

The effect of the amount of training data

The first experiment again examines the effect of increasing training set size on the performance of each method. The result are shown in Figure 3-4. We can see that OCT and OCT-H have very similar performance in nearly all cases; the FDR of OCT-H is slightly higher than OCT for lower training set sizes, indicating the increased power is leading to trees with slightly more noise when not enough data is available. However, this quickly vanishes as the number of points grows and OCT-H even has a slightly lower FDR for the largest datasets, roughly three times smaller than CART. For very small training set sizes, boosting has significantly higher out-of-sample accuracy than all other methods, but past 500 training points the highest accuracies are shared by OCT, OCT-H and boosting. Random forests are weaker than these three methods, but still offer an advantage over CART that diminishes with increasing

Figure 3-5: Synthetic experiments showing the effect of number of features.



training set size.

The effect of dimensionality

The second experiment shows the performance of the methods as we change the number of features in the data. The results are presented in Figure 3-5. We can see that OCT, OCT-H and boosting all have roughly the same out-of-sample accuracy at all dimensions, while the accuracy of random forests quickly falls as the dimension increases. The TDR of OCT-H is initially similar to OCT, and then falls to the same level as CART at the higher numbers of features. Similarly, the FDR starts close to OCT, then quickly rises past CART to 50% at the highest. This is unsurprising because the true splits contain just a single feature, and so any hyperplane split with more than one feature is deemed incorrect, and as the number of features increases we would expect to see more splits involving multiple features because there are many more such splits to consider.

The effect of the number of classes

The third experiment varies the number of classes in the ground truth tree, and the results are shown in Figure 3-6. Again we see near-identical performance for OCT and OCT-H in all measures. For binary problems, boosting has a high accuracy, but this accuracy decreases rapidly as the number of classes is increased, while the accuracy

Figure 3-6: Synthetic experiments showing the effect of number of classes.

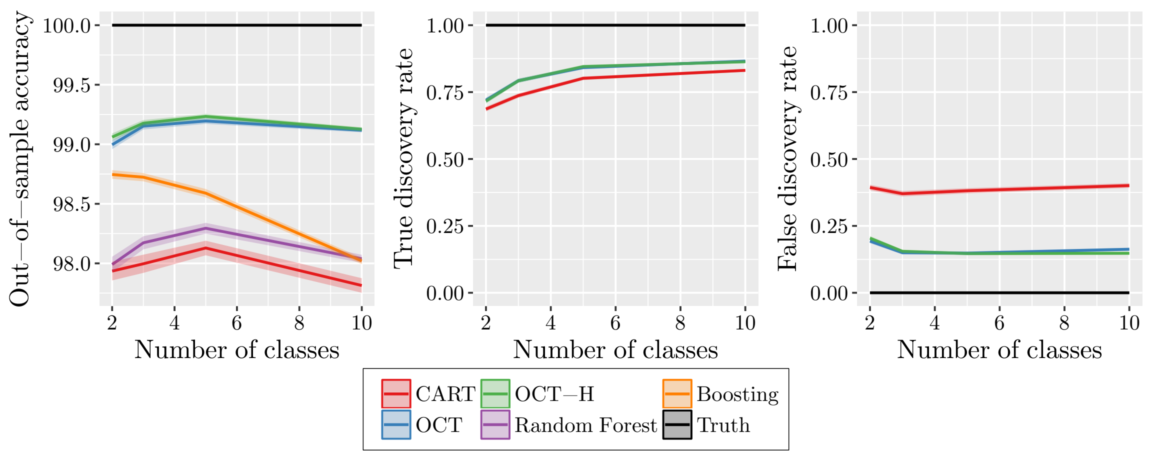
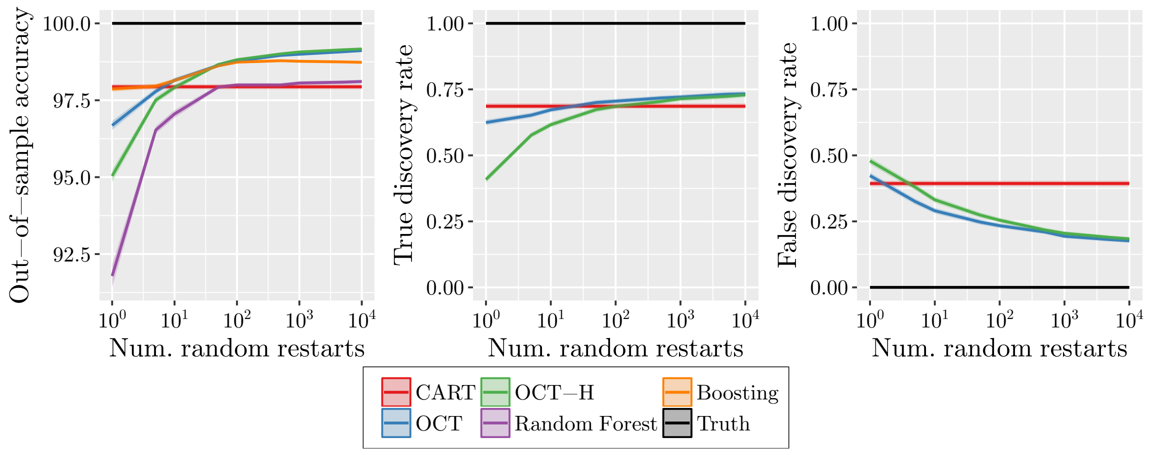


Figure 3-7: Synthetic experiments showing the effect of number of random restarts.

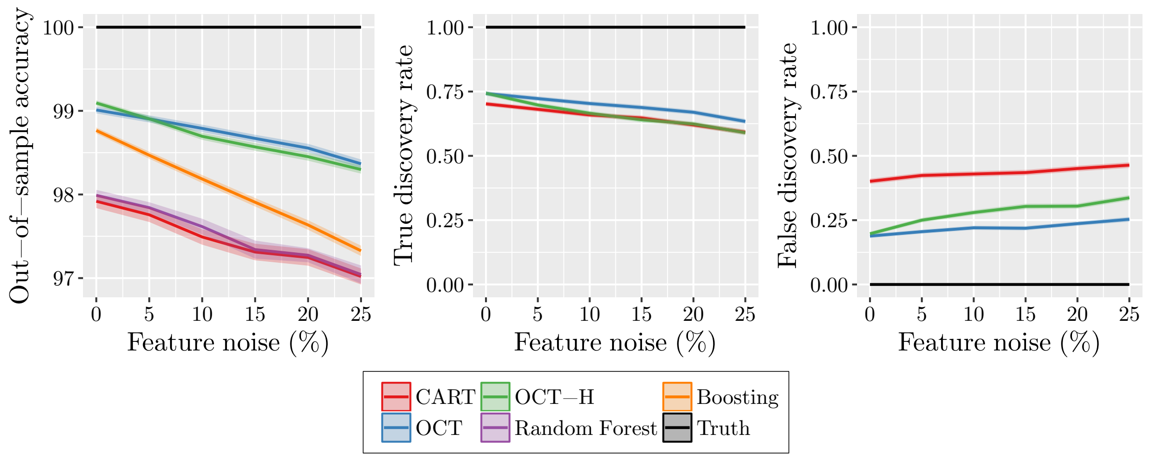


of OCT and OCT-H remains relatively constant. This would seem to indicate that boosting is not well-suited to problems with many classes in the same way as CART. The accuracy of random forests also remains relatively constant as the number of classes increases, but at a significantly lower level, closer to CART than to OCT and OCT-H.

The effect of the number of random restarts

The fourth experiment examines the effect of the number of random restarts in the local search on the performance of OCT and OCT-H. For comparison, we also set the number of trees used in random forests and boosting to the same number, so that each method is training the same number of trees. Figure 3-7 shows the results.

Figure 3-8: Synthetic experiments showing the effect of feature noise.

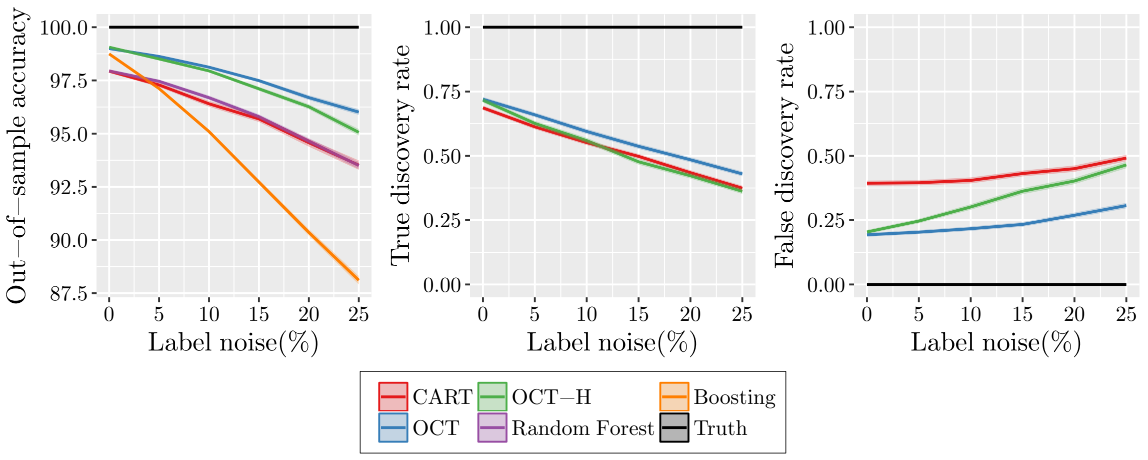


We see that OCT-H performs worse than OCT in all measures at lower numbers of restarts, but largely catches up to OCT between 100–1000 restarts. This showcases that the increased complexity and power of OCT-H means it is not as stable as OCT until a moderate number of restarts are used. The accuracy of boosting at around 100 trees is a similar level to OCT and OCT-H, but as more trees are added it does not continue to improve, unlike OCT and OCT-H. Random forests level out matching the performance of CART at around 100 restarts. These results demonstrate that the Optimal Tree methods are able to make use of additional trees in an effective manner, whereas ensemble methods appear to reach a peak and largely do not improve from there.

The effect of noise in the training data

The fifth experiment measures the effect of the amount of noise in the features of the data. Like the corresponding experiment in Section 2.5, we added uniform noise to $f\%$ of the training points. The results are shown in Figure 3-8. We can see that all methods decline in accuracy as the amount of feature noise increases, but boosting decreases faster than the other methods. With no noise, boosting performs comparably to OCT and OCT-H, but at the highest levels of noise it is significantly weaker than both, comparable to CART and random forests. The TDR of OCT and OCT-H is very similar, while OCT-H has a slightly worse FDR than OCT, which

Figure 3-9: Synthetic experiments showing the effect of label noise.



we might expect since the feature noise could make it more difficult to identify the correct split out of the many more possible hyperplane splits.

The final experiment considers the effect of adding label noise to the training data. This is again achieved by flipping the labels of a random $f\%$ of the points. Figure 3-9 shows the results. We can immediately see that boosting is very sensitive to the presence of label noise. When there is no noise, it performs strongest along with OCT and OCT-H, but at around 5% label noise it is tied for weakest with CART, before falling significantly further as more noise is added. Admittedly, this highest level of noise involves 25% of the points being labeled incorrectly, but OCT and OCT-H are still able to achieve 95% accuracy out-of-sample in this case, demonstrating they are able to cope with such high levels of noise. As we saw in the previous test with feature noise, the FDR of OCT-H suffers as the amount of label noise increases, which is again likely due to the increased difficulty in exactly identifying the correct splits in such a noisy environment.

Summary for ground truth trees with parallel splits

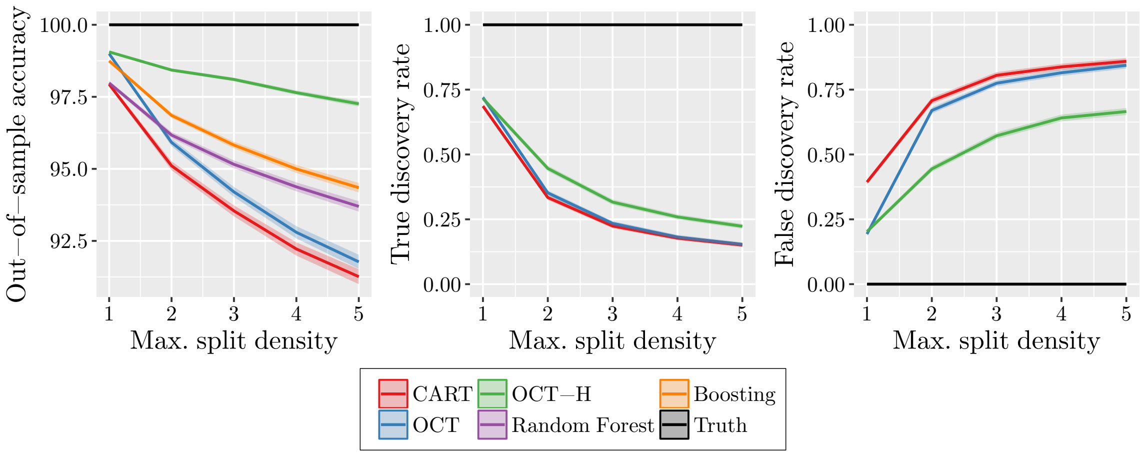
We now summarize the results of these repeated experiments with parallel ground truth trees from Section 2.5. In all cases, OCT and OCT-H had near-identical results, indicating that the additional power of OCT-H is not leading it to overfit with more complicated trees, and instead it is learning the structure in the data with ability similar to the parallel-only trees of OCT. The main exception was in the FDR of OCT-H as the number of features became very large, where the lack of training data relative to the number of possible hyperplane splits means it is very easy for OCT-H to find splits that are not the same as in the true tree. We note that the accuracy of the trees did not suffer in this regime, just the FDR. This indicates that we should be cautious in interpreting hyperplane trees when the number of points is small relative to the dimension of the points.

We found that boosting was typically the best of the other methods, with a large advantage over random forests, which in turn offered a small improvement over CART. However, despite its strength overall, boosting suffered in a few key settings, namely as the number of classes to predict increases, and in the presence of either feature or label noise. In all of these settings, OCT and OCT-H demonstrated significantly better capacity to deal with the increasing problem difficult. Overall, OCT and OCT-H were able to match or beat the performance of boosting in all tests, indicating that we need not sacrifice interpretability to achieve the highest accuracy if we have reason to believe the data has an underlying parallel tree structure.

Ground truth trees with hyperplane splits

The second set of experiments measures the performance of the various methods on datasets that were generated from ground truth trees with hyperplane splits. To control the generation of these hyperplane ground truth trees, we use a new parameter called the *maximum split density*, which is the maximum number of features allowed to be used in any split of tree. To generate a random hyperplane split in the tree, we generate a random number between one and the maximum split density, and use this as the number of feature to include in the split. We then randomly select which of the features to use, and generate a random value for each to use in the split. This gives a tree with hyperplanes that have a variety of split densities, which is important because it allows us to examine whether OCT-H maintains sparsity and only uses as many features as are required by each split, or whether it simply overfits with very dense splits.

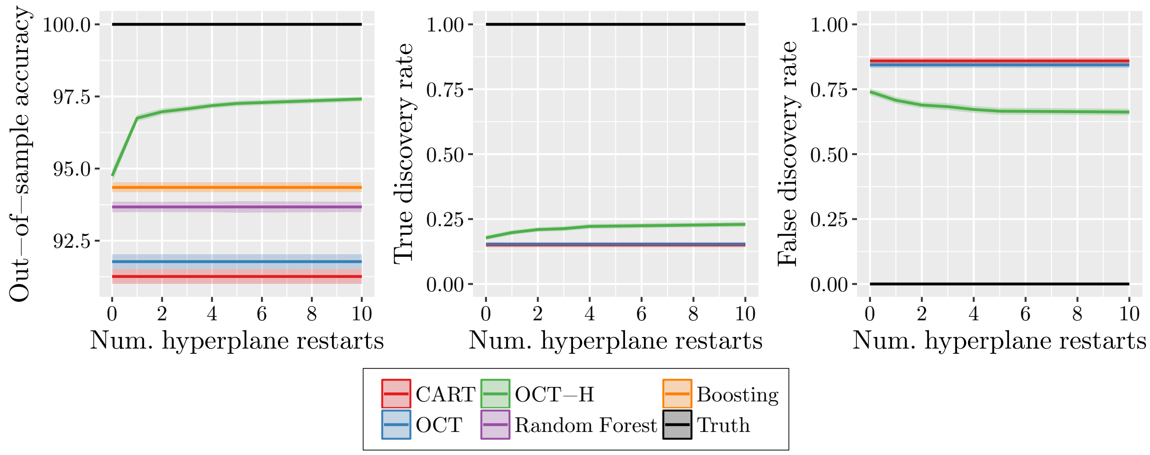
Figure 3-10: Synthetic experiments showing the effect of split density.



The effect of split density

The first experiment measures the effect of the maximum split density in the ground truth tree on the performance of each method. The results are shown in Figure 3-10. When the maximum split density is 1, the ground truth trees just have parallel splits, so the performances are similar to the earlier experiments. We see that the out-of-sample accuracy of all methods decreases as the maximum split density increases, which we would expect because the problem is becoming significantly more difficult. CART and OCT have the largest decrease in accuracy with the split density, indicating that, unsurprisingly, parallel trees are not well-suited to learning a hyperplane tree structure. Despite this, OCT still offers an advantage over CART, showing the power of the optimization approach. Random forests and boosting also have a large decrease in accuracy as the maximum split density increases, but they both offer significant improvements over CART and OCT. This would seem to indicate that the aggregation approach used by these methods allows them to better approximate the hyperplane structure in the data, despite their use of parallel splits only. Finally, OCT-H has the highest accuracy by a significant margin, and only loses a small amount of accuracy as the maximum density increases, showing that it has enough power to learn well the truth in the data. OCT-H also significantly outperforms CART and OCT in both TDR and FDR, showing that it is able to learn the true hyperplane splits in the data, although the performance of OCT in both these

Figure 3-11: Synthetic experiments showing the effect of number of random hyperplane restarts.



measures falls as the maximum density increases due to the increased difficulty of the problem with respect to the fixed amount of data available.

The effect of the number of random hyperplane restarts

The next experiment considers the effect of the number of random hyperplane restarts H on the performance of OCT-H. Figure 3-11 shows the performance of the OCT-H method as H increases, with the maximum split density in the ground truth tree being fixed at 5. We see that initially OCT-H has a similar accuracy to boosting and random forests when $H = 0$, and then there is a very significant increase in accuracy when H increases to 1. This indicates that simply perturbing the best parallel split does not find the best hyperplane splits, and it is much more powerful to perturb a randomized hyperplane split. As the number of hyperplane restarts is further increased, the performance improves slightly, but not nearly as significantly as the jump from $H = 0$ to $H = 1$. There is clear evidence that OCT-H should always be used with $H \geq 1$, and the exact value used should depend on the amount of time available. We use these results as guidance to recommend that OCT-H is used with $H = 5$ to $H = 10$ in practice, as this is sufficient to capture most of the benefit of additional restarts without increasing the runtime too greatly. For specific problems, it may be sufficient to use a smaller value of H (e.g. $H = 2$ is probably good enough in this synthetic example), but one should be wary of setting H too small and losing accuracy, hence our suggestion to start with H between 5 and 10.

Summary for ground truth trees with hyperplane splits

Both experiments involving ground truth trees with hyperplane splits demonstrated that OCT-H is able to perform well and learn the true tree structure even when the splits in the tree are dense. The ensemble methods (random forests and boosting) improved upon the single parallel-tree methods (OCT and CART), but were unable to match the performance of OCT-H. This is strong evidence that the aggregation approach of these methods is significantly less powerful for modeling non-parallel splits when compared to just modeling these splits directly.

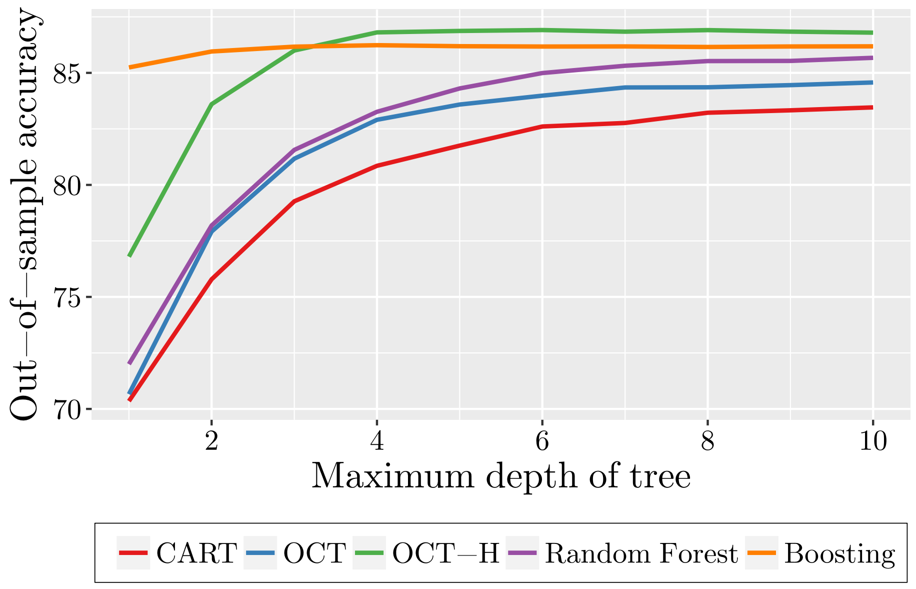
We also saw that there was a significant increase in performance for OCT-H when changing from $H = 0$ hyperplane restarts to $H = 1$. The improvements for higher H were much less dramatic, yet still significant, and so we recommend that a value from 5–10 typically be used.

As seen with the parallel-split ground truth trees, we found that OCT-H beat the performance of the ensemble methods in all cases with hyperplane-split ground truth trees, demonstrating that we can achieve the highest accuracy with a model that directly models the true structure of the underlying data, and moreover, in doing so we generate a resulting model that is directly interpretable.

3.5 Experiments with Real-World Datasets

In this section, we benchmark the performance of Optimal Classification Trees with Hyperplanes against both CART and standard Optimal Classification Trees on the same sample of real-world datasets as Section 2.6. We also provide wider context for these results by conducting comparisons with Random Forests and Gradient Boosting, two tree-based classification methods that achieve state-of-the-art accuracy.

Figure 3-12: Mean out-of-sample accuracy for each method across all 60 datasets.



Experimental Setup

We used the same experimental setup and sample of 60 datasets as in Section 2.6 for these experiments. In addition to running CART and OCT as described in Section 2.6, we ran OCT-H with $R = 100$ random restarts and $H = 10$ hyperplane restarts, using Algorithm 2.4 to tune the values of the complexity parameter α and the depth of the tree. We also ran Random Forests and Gradient Boosting Trees using the same implementations as described in Section 3.4.

Results and Discussion

Figure 3-12 shows the mean out-of-sample accuracies of all methods on the sample of 60 datasets according to the depth of the tree used.

First, we will compare OCT-H and the parallel split methods OCT and CART in order to understand and quantify the additional power of hyperplane splits. We can see that OCT-H delivers significantly better out-of-sample accuracy than CART and OCT at all depths. OCT-H with trees of depth 2 outperforms CART at all depths, and performs about the same as OCT at depth 5. OCT-H at depth 3 significantly outperforms OCT at all depths. We also see that the accuracy of OCT-H stops increasing after depth 4. These results reinforce the assertion in Section 3.3 that trees with hyperplane splits can achieve comparable (or indeed better) accuracies to those with parallel splits with much smaller trees, and therefore are not necessarily strictly less interpretable.

Next, we include random forests and boosting in the comparison, in order to place the accuracy results of OCT and OCT-H in a wider context by comparing them to two methods that give state-of-the-art accuracies. We can see that OCT and random forests perform roughly the same up to depth 4, after which random forests have a small advantage. Boosting is the strongest method at depths 1 and 2, but OCT-H catches up at depth 3 and then has a small edge over boosting at depths 4 and larger. When the depth is 10 and is effectively unconstrained, we see that OCT-H is the strongest method with a small advantage over boosting, which in turn has a small advantage over random forests. There is then a larger gap down to OCT and then CART.

Based on these aggregate results across all 60, it seems that OCT-H likely offers a small advantage over random forests and has comparable performance to boosting methods. We will now investigate these effects more extensively.

Table 3.1 shows both the accuracy and performance rank of each method on each of the datasets. These ranks allows us to understand how the methods perform relative to one another on each individual dataset rather than just in aggregate. The average rank at the bottom gives the mean rank of each method across all datasets. These ranks give evidence that CART is the weakest-performing method by a significant margin, followed by OCT, which is not surprising as these methods are limited to a single tree with parallel splits only. Boosting and random forests have the best average rank, followed very closely by OCT-H.

Next, we conduct pairwise tests between the methods to determine which have statistically significant differences in performance. Table 3.2 shows the results of these comparisons. We follow the approach recommended by [40] and [50] for comparing the results of multiple methods on multiple datasets. For each pair of methods, we test significance using the Wilcoxon signed-rank test, and the resulting p-values are

Table 3.1: Performance results for classification methods (with maximum depth 10) on each of the 60 datasets. For each method, we show the out-of-sample accuracy (%) followed by the method's rank on the dataset in parentheses. A rank of 1 indicates the method had the best performance on a dataset and a rank of 5 indicates the method performed worst.

Dataset	CART	OCT	OCT-H	RF	Boosting
acute-inflammations-1	95.33 (5)	100.00 (1.5)	98.00 (3.5)	100.00 (1.5)	98.00 (3.5)
acute-inflammations-2	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	98.67 (5)
balance-scale	78.09 (5)	79.36 (4)	89.94 (2)	85.22 (3)	91.21 (1)
banknote-authentication	98.02 (5)	98.89 (4)	99.71 (1)	98.95 (3)	99.48 (2)
blood-transfusion	77.22 (5)	78.07 (3)	78.61 (1)	77.65 (4)	78.50 (2)
breast-cancer-diagnostic	90.91 (5)	92.73 (4)	94.83 (2)	94.55 (3)	96.08 (1)
breast-cancer-prognostic	75.51 (1.5)	75.51 (1.5)	74.29 (3)	74.29 (4)	70.20 (5)
breast-cancer	94.97 (5)	95.09 (4)	97.19 (2)	97.78 (1)	96.37 (3)
car-evaluation	91.25 (4)	92.41 (3)	97.82 (1)	86.44 (5)	96.94 (2)
chess-king-rook-vs-king-pawn	98.90 (4)	99.32 (3)	99.40 (1)	97.00 (5)	99.40 (2)
climate-model-crashes	91.26 (5)	92.30 (3)	93.04 (2)	92.30 (4)	94.22 (1)
congressional-voting-records	98.62 (2.5)	98.62 (2.5)	97.24 (5)	98.62 (2.5)	98.62 (2.5)
connectionist-bench-sonar	69.23 (5)	75.00 (4)	77.31 (3)	84.62 (1)	84.62 (2)
contraceptive-method-choice	53.22 (4)	53.17 (5)	55.93 (1)	53.66 (3)	55.12 (2)
credit-approval	87.24 (1)	86.75 (3)	86.01 (5)	87.24 (2)	86.63 (4)
cylinder-bands	66.09 (5)	67.54 (4)	72.75 (2)	75.36 (1)	72.46 (3)
dermatology	95.06 (5)	95.28 (4)	96.18 (3)	98.20 (1)	96.40 (2)
echocardiogram	72.00 (5)	73.33 (4)	76.00 (1)	74.67 (2.5)	74.67 (2.5)
fertility	87.20 (1)	86.40 (2.5)	82.40 (5)	86.40 (2.5)	85.60 (4)
haberman-survival	73.51 (1)	73.25 (2)	72.73 (3)	70.65 (5)	71.17 (4)
hayes-roth	75.76 (5)	78.18 (3.5)	78.18 (3.5)	80.00 (1)	79.39 (2)
heart-disease-cleveland	57.60 (3)	55.47 (5)	56.00 (4)	58.93 (1)	58.13 (2)
hepatitis	84.00 (2)	82.00 (3)	78.00 (5)	87.00 (1)	80.00 (4)
hill-valley-noise	55.76 (4)	56.56 (2)	78.15 (1)	56.16 (3)	53.64 (5)
hill-valley	49.80 (5)	52.58 (3)	98.41 (1)	51.26 (4)	57.22 (2)
image-segmentation	77.74 (5)	86.79 (4)	87.92 (3)	94.72 (1)	90.19 (2)
indian-liver-patient	71.59 (1)	71.17 (2)	70.90 (3)	70.34 (4)	70.07 (5)
ionosphere	88.28 (4)	91.26 (3)	86.67 (5)	94.71 (1)	93.56 (2)
iris	93.51 (5)	94.59 (3)	94.59 (3)	95.14 (1)	94.59 (3)
magic-gamma-telescope	84.91 (4)	84.66 (5)	86.88 (2)	86.37 (3)	88.38 (1)
mammographic-mass	81.74 (2.5)	80.48 (5)	81.16 (4)	81.74 (2.5)	82.80 (1)
monks-problems-1	74.84 (5)	87.74 (3)	98.06 (1)	78.71 (4)	87.74 (2)
monks-problems-2	59.53 (3.5)	60.00 (2)	88.84 (1)	48.84 (5)	59.53 (3.5)
monks-problems-3	94.19 (1)	92.90 (3.5)	93.55 (2)	92.90 (3.5)	90.97 (5)
mushroom	99.96 (5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)
optical-recognition	88.63 (4)	88.02 (5)	91.94 (3)	97.53 (1)	97.38 (2)
ozone-level-detection-eight	93.06 (4)	92.97 (5)	93.19 (3)	93.58 (2)	93.88 (1)
ozone-level-detection-one	96.49 (4)	96.62 (3)	96.23 (5)	96.67 (2)	96.75 (1)
parkinsons	87.76 (2)	86.94 (4.5)	87.76 (3)	88.57 (1)	86.94 (4.5)
pen-based-recognition	96.15 (4)	95.54 (5)	97.75 (3)	98.58 (2)	99.24 (1)
pima-indians-diabetes	72.29 (5)	73.02 (4)	73.13 (3)	74.17 (2)	75.42 (1)
planning-relax	71.11 (1.5)	71.11 (1.5)	70.22 (3)	70.22 (4)	68.44 (5)
qsar-biodegradation	82.36 (5)	83.50 (4)	85.63 (3)	88.29 (1)	88.06 (2)
seeds	89.43 (4)	88.30 (5)	91.70 (1)	90.94 (2)	90.57 (3)
seismic-bumps	92.91 (4)	93.22 (2.5)	92.79 (5)	93.28 (1)	93.22 (2.5)
skin-segmentation	99.72 (5)	99.89 (3)	99.93 (2)	99.83 (4)	99.95 (1)
soybean-small	100.00 (3)	100.00 (3)	100.00 (3)	100.00 (3)	100.00 (3)
spambase	91.47 (5)	93.17 (4)	94.21 (2)	93.97 (3)	95.93 (1)
spect-heart	61.00 (4.5)	62.00 (3)	61.00 (4.5)	71.00 (2)	73.00 (1)
spectf-heart	72.00 (4)	76.00 (3)	62.00 (5)	77.00 (2)	79.00 (1)
statlog-german-credit	72.00 (4)	70.96 (5)	72.08 (3)	73.92 (2)	75.12 (1)
statlog-landsat	85.28 (5)	85.73 (4)	87.30 (3)	89.83 (2)	91.22 (1)
teaching-assistant	56.76 (4)	63.24 (2)	63.78 (1)	53.51 (5)	60.00 (3)
thoracic-surgery	84.79 (2.5)	84.79 (2.5)	84.44 (4)	85.13 (1)	84.10 (5)
thyroid-disease-ann	99.75 (1)	99.72 (2)	99.66 (3.5)	99.55 (5)	99.66 (3.5)
thyroid-disease-new	92.83 (5)	93.96 (4)	95.47 (3)	97.36 (1)	96.23 (2)
tic-tac-toe-endgame	94.06 (4)	93.97 (5)	95.73 (3)	98.83 (2)	99.08 (1)
wall-following-robot-2	100.00 (2)	100.00 (2)	100.00 (2)	99.97 (4.5)	99.97 (4.5)
wall-following-robot-24	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	100.00 (2.5)	99.97 (5)
wine	84.89 (5)	94.22 (4)	95.11 (3)	98.22 (1)	97.33 (2)
Average rank	3.758	3.375	2.775	2.533	2.558

Table 3.2: Pairwise significance tests for performance of classification methods (with maximum depth 10) across 60 datasets. The comparisons are presented in order of significance. For each comparison, the strongest-performing method is both bolded and stated first in the comparison. The p-values are found using Wilcoxon signed-rank test, and the adjusted p-values are calculated using the Holm-Bonferroni method to account for multiple comparisons. All results above the dividing line are significant at the 95% level.

Comparison	p-value	Adjusted p-value
Boosting vs. CART	$\sim 10^{-5}$	0.0003
Random Forest vs. CART	$\sim 10^{-4}$	0.0005
OCT-H vs. CART	0.0003	0.0021
Boosting vs. OCT	0.0005	0.0034
OCT-H vs. OCT	0.0023	0.0135
OCT vs. CART	0.0025	0.0135
Random Forest vs. OCT	0.0029	0.0135
OCT-H vs. Boosting	0.1507	0.4522
OCT-H vs. Random Forest	0.1932	0.4522
Boosting vs. Random Forest	0.6064	0.6064

then adjusted using the Holm-Bonferroni method [62] to control the familywise error rate. The comparisons are presented in order of significance. We see that CART is statisically significantly weaker than OCT, and both CART and OCT are weaker than OCT-H, random forests, and boosting. The comparisons find no statistically significant difference in performance between OCT-H, boosting, or random forests.

Together, these computational experiments with real datasets offer strong evidence that OCT-H is competitive with the state-of-the-art classification methods in practical settings, with comparable accuracies and no statistically significant difference in performance to random forests and boosting. The key difference is that OCT-H achieves this performance with just a single decision tree, and so the resulting classifier is significantly more interpretable than the other methods.

3.6 Conclusions

In this chapter, we extended our approach for constructing Optimal Classification Trees to also consider making splits that were not restricted to be parallel to the axes.

Prior attempts at creating decision trees with such splits were both not tractable and did not lead to significant improvements over normal decision trees, and thus have not seen practical use in applications. The increased difficulty in finding these hyperplane splits is because there are an exponential number of possible hyperplane splits to search over rather than a linear number in the axes-parallel case.

We made minor modifications to the MIO formulation for Optimal Classification Trees to allow for these hyperplane splits, giving a formulation that can be solved to find the globally optimal classification tree with hyperplane splits. As for the parallel splits, we found that the MIO formulation did not scale well to practical problem sizes, and so we adapted the local search procedure to also optimize hyperplane splits. The trees constructed using this approach are significantly stronger than those found using the MIO formulation with a large time limit, and are found in a fraction of the time.

We conducted extensive experiments with both synthetic and real-world datasets in order to compare the performance of our optimal tree methods against the state-of-the-art in classification. The experiments with synthetic data demonstrated that our optimal trees are significantly stronger than random forests and boosted trees in cases where the true underlying structure in the data is a tree. Moreover, the experiments with 60 real-world datasets offer strong evidence that OCT-H also has comparable performance to random forests and boosted trees in practical settings.

Together these results demonstrate that our optimal tree methods can deliver state-of-the-art performance for classification problems without sacrificing the key interpretability advantage of a single decision tree.

Chapter 4

Optimal Regression Trees with Constant Predictions

In the previous chapters, we have focused exclusively on training trees for classification problems. Another central problem in statistics and machine learning is regression, where the outcomes are no longer discrete categories, but instead are continuous-valued.

Regression trees are harder to train compared to classification trees. In the leaf of a classification tree, it is simple and computationally cheap to calculate the optimal prediction using the majority rule, and this is clearly the best prediction rule to use. However, for a regression tree there are many options for predicting the outcomes of the points in the leaf, which range in computational cost, simplicity, and interpretabilty. As a result, it is not clear which prediction rule is best for regression problems, and for this reason regression trees have historically received less attention in the literature compared to classification trees.

The most common prediction rule for regression trees is to use a constant prediction for all points in the leaf, which is found by calculating the mean outcome of the points in the leaf. This approach is used by CART and other methods [25, 72, 63]. Training a tree with constant predictions in the leaves represents fitting a piece-wise constant function to the training data. Constant predictions are chosen primarily for computational reasons, since calculating the mean outcome can be done very efficiently. The predictions are not as accurate as other regression methods, and therefore we might require deep trees to achieve a good level of accuracy on the dataset.

In this chapter, we apply the Optimal Trees methodology for classification problems from Chapters 2 and 3 to the regression problem, yielding a procedure for generating Optimal Regression Trees with constant predictions in each leaf. Our goal is to investigate the effects of solving the regression tree problem optimally with constant predictions in each leaf, as such trees are directly comparable to the regression trees currently used in practice. We will consider trees with more sophisticated prediction functions in each leaf in Chapter 5.

4.1 Review of Regression Tree Methods

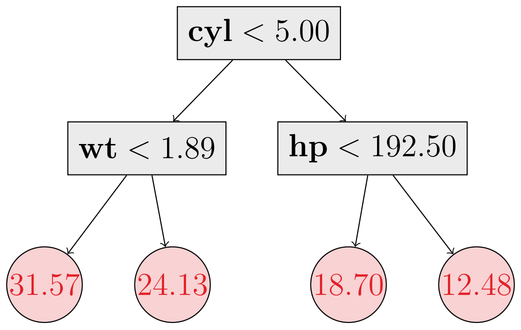
In regression problems, we are supplied training data $(\mathbf{X}, \mathbf{y})$, containing n observations $(\mathbf{x}_i, y_i)$, $i \in [n]$, each with p features $\mathbf{x}_i \in \mathbb{R}^p$ and an outcome $y_i \in \mathbb{R}$. As with the classification problem, we will assume without loss of generality that the training values have been normalized to the unit interval, so that each $\mathbf{x}_i \in [0, 1]^p$.

Regression trees take a near-identical structure to classification trees. The only difference is that the prediction in each leaf is for the continuous outcome as opposed to a discrete class label. In this chapter we deal only with regression trees that have constant prediction functions in each leaf, so each point in a leaf will receive the same predicted outcome, just like in classification.

Figure 4-1 shows an example of a regression tree trained on the "mtcars" dataset [60]. This dataset comes from *Motor Trend* magazine, and the goal is to predict fuel consumption of automobiles based on variables describing the design and performance of the automobile. We can see that the tree gives us an interpretable and intuitive explanation of which factors lead to higher fuel consumption.

The tree growing procedure of CART is the same for classification as for regression, except that we use a different loss function when selecting a split. In each leaf, it calculates the squared loss that results from using the mean outcome in the leaf as the final prediction, because the mean is the constant prediction that minimizes the

Figure 4-1: CART regression tree predicting fuel consumption (in miles per gallon) of automobiles using the "mtcars" dataset. "cyl" denotes the number of cylinders in the engine, "wt" denotes the weight of the vehicle (in 1000 pounds), and "hp" denotes the gross horsepower.



squared loss on the points in the leaf. It then selects the split that results in the lowest overall squared loss, and then repeats recursively. Like classification, it stops when either the minimum bucket size N_{min} is reached, or when no further improvement in error is possible.

Again mirroring classification, the final step in the process is pruning the tree to limit the complexity. This is achieved in exactly the same way as classification by using the complexity parameter to control the tradeoff between accuracy and complexity. For regression, the squared loss is used for both the training and pruning phases, rather than using a different loss function for each like in classification.

Based on this procedure, we can see that CART also aims to solve Problem (2.1) when constructing regression trees, except in this case $R(\mathbb{T})$ represents the mean-squared error of the tree on the training data, rather than the misclassification error.

4.2 ORT with Constant Predictions via MIO

In this section, we follow the example of Optimal Classification Trees from Chapters 2 and 3 and formulate the task of constructing the globally optimal regression tree as an MIO problem.

First, we note that the majority of the formulation for Optimal Classification Trees can simply be reused for training regression trees. The only changes that need to be made are to the variables and constraints that determine the objective. For the objective, we will consider the absolute and squared losses, the two most-commonly used functions in regression.

At each leaf node t in the tree, we will make the same constant prediction given by the continuous variable $\beta_{0t} \in \mathbb{R}$. We then use the variables f_i to denote the fitted value that is predicted by the regression tree for each point i. This can be calculated using the following expression:

$$f_i = \sum_{t \in \mathcal{T}_L} \beta_{0t} z_{it}, \quad \forall i \in [n]. \tag{4.1}$$

Since only one of the $z_{it} = 1$, indicating that the tth leaf contains point i, only the term corresponding to the prediction function at leaf t will remain, and the corresponding prediction will be assigned for point i. This expression is non-linear in the variables, however we can linearize it as follows:

$$-M_f(1 - z_{ik}) \leq f_i - \beta_{0t} \leq M_f(1 - z_{ik}), \quad \forall i \in [n],\ t \in \mathcal{T}_L, \tag{4.2}$$

where M_f is a sufficiently large constant. We can see that if $z_{it} = 1$, this forces $f_i = \beta_{0t}$, and otherwise the constraint disappears. In order to specify a value for M_f, we need to identify the largest possible value of $f_i - \beta_{0t}$. We observe that at optimality, the fitted values f_i will lie in the range of $\mathbf{y}$, and therefore so will the predictions β_{0t}. This means that we can select $M_f = \max_i y_i - \min_i y_i$ without affecting the feasibility of any optimal solutions.

We now want to calculate the loss for each point i, denoted by the variable L_i, based on the difference between the fitted value f_i and actual value y_i.

First, we consider the absolute loss. We can set L_i as the absolute loss for the ith point as follows:

$$L_i = |f_i - y_i|, \quad \forall i \in [n], \tag{4.3}$$

which can be linearized to give

$$L_i \geq +f_i - y_i, \quad \forall i \in [n], \tag{4.4}$$

$$L_i \geq -f_i + y_i, \quad \forall i \in [n], \tag{4.5}$$

This linearization is valid under the knowledge that we will be minimizing the values of L_i as the objective, as in this case L_i will either take the value of $f_i - y_i$ or $-f_i + y_i$.

We can also use the squared loss. In this case, we set L_i as follows:

$$L_i \geq (f_i - y_i)^2, \quad \forall i \in [n], \tag{4.6}$$

which is a quadratic constraint. Similar to the absolute loss, this inequality is valid under the knowledge that we are minimizing L_i in the objective, as the constraint will become tight at optimality. Quadratic constraints, while more complex than linear constraints, are supported in recent versions of the state-of-the-art MIO solvers, and so the problem with squared loss can be solved with the same solvers as the other formulations.

We want to minimize the total loss across all our predictions plus a penalty on tree complexity, which for trees with parallel splits gives:

$$\min \quad \frac{1}{\hat{L}} \sum_{i=1}^{n} L_i + \alpha \cdot C, \tag{4.7}$$

where we have normalized the total loss by the baseline loss $\hat{L}$ obtained by making a single prediction for the entire training set. As for classification, this normalization is done to make the effect of α independent of the training set size.

The complete MIO formulation can be obtained by using these new constraints and variables to replace the classification elements either in Problem (2.24) for parallel splits or in Problem (3.3) for hyperplane splits. For instance, the following is the complete *Optimal Regression Trees* (ORT) model with squared loss:

$$
\begin{aligned}
\min \quad & \frac{1}{\hat{L}} \sum_{i=1}^{n} L_i + \alpha \cdot C && && (4.8)\\
\text{s.t.} \quad & L_i \geq (f_i - y_i)^2, && \forall i \in [n], \\
& f_i - \beta_{0t} \geq -M_f(1 - z_{ik}), && \forall i \in [n],\ t \in \mathcal{T}_L, \\
& f_i - \beta_{0t} \leq +M_f(1 - z_{ik}), && \forall i \in [n],\ t \in \mathcal{T}_L, \\
& C = \sum_{t \in \mathcal{T}_B} d_t, \\
& \mathbf{a}_m^{\mathsf{T}} \mathbf{x}_i \geq b_m - (1 - z_{it}), && \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t), \\
& \mathbf{a}_m^{\mathsf{T}} (\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), && \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t), \\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, && \forall i \in [n], \\
& z_{it} \leq l_t, && \forall t \in \mathcal{T}_L, \\
& \sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, && \forall t \in \mathcal{T}_L, \\
& \sum_{j=1}^{p} a_{jt} = d_t, && \forall t \in \mathcal{T}_B, \\
& 0 \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B, \\
& d_t \leq d_{p(t)}, && \forall t \in \mathcal{T}_B \setminus \{1\}, \\
& z_{it}, l_t \in \{0, 1\}, && \forall i \in [n],\ k \in [K],\ t \in \mathcal{T}_L, \\
& a_{jt}, d_t \in \{0, 1\}, && \forall j \in [p],\ t \in \mathcal{T}_B.
\end{aligned}
$$

The size of these regression tree formulations is similar to their classification counterparts. As mentioned earlier, the formulations using the absolute loss will remain linear MIO problems, whereas the squared loss will lead to a quadratic MIO, which are marginally harder to solve that their linear counterparts, but remain solvable with commercial MIO solvers like Gurobi. Together, these facts mean that these MIO formulations will have similar solution times to the corresponding classification problems.

4.3 ORT with Constant Predictions via Local Search

In this section, we modify the local search heuristic developed in Sections 2.3 and 3.2 to generate regression trees, allowing us to generate Optimal Regression Trees both without (ORT) and with (ORT-H) hyperplane splits.

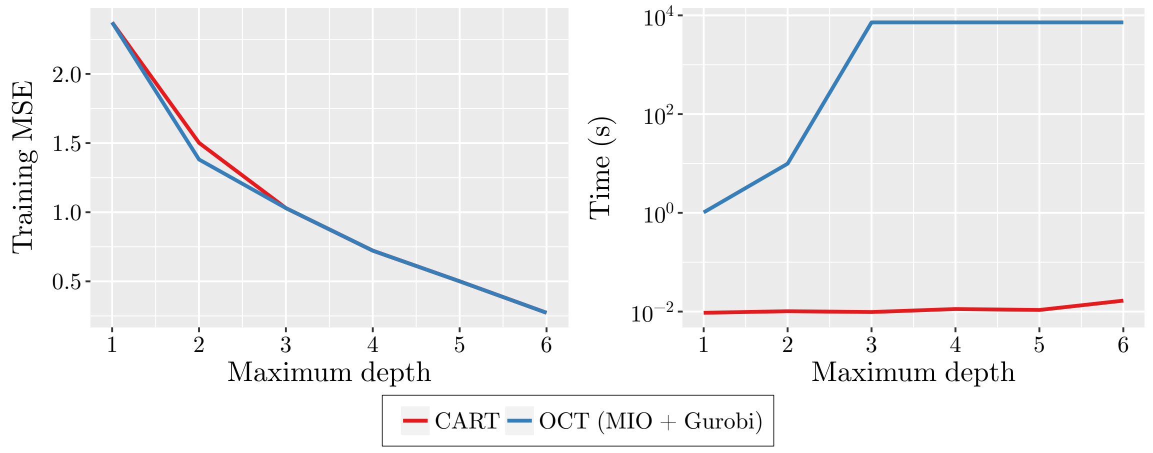
The MIO formulations for the ORT and ORT-H problems in Section 4.2 share the same problems with scaling as the formulations for classification trees. Driven by the number of training points and the depth of the tree, the size of the MIO formulation, and in particular the number of binary variables, grows very fast and quickly leads to the MIO formulation becoming intractable.

Figure 4-2 demonstrates the scaling issues of the MIO approach. We trained CART and ORT trees on the "Hybrid Vehicle Prices" dataset made available in [78], which is a small dataset with $n = 152$ and $p = 3$ (after excluding a single categorical feature). We plot the training error and running time for both methods against the depth of the tree being trained. The MIO problem was limited to 2 hours running time. We can see that both methods find the same solution at depth 1, with MIO providing a proof of optimality for this solution. At depth 2, MIO again solves the problem to optimality, and improves upon the error of the CART solution. At depths 3–6 however, the MIO solution remains unchanged from the CART warm start after the 2 hour timelimit has elapsed. Similar to classification, we believe it is likely that the CART solutions can actually be improved, but that the MIO solver is simply unable to make progress within the timelimit. This motivates a search for more effective solution approaches.

Recall that for classification problems, we observed that once the points were assigned to leaves in the tree, the resulting predictions and accuracies were closed-form solvable by simply assigning the majority class in each leaf. This motivated our local-search heuristic where we modify the tree one split at a time and evaluate the objective function in closed-form after each modification.

Regression trees also share this characteristic of closed-form solvability once the leaf assignment is fixed. For squared loss, we simply predict the mean of the labels in

Figure 4-2: Training mean squared error and running time for each method on "Hybrid Vehicle Prices" dataset.



each leaf, and for absolute loss the median. This leads us to develop a similar local-search heuristic for regression trees by simply applying the same search procedure and substituting a different method for evaluating the loss in closed-form. The full local-search procedure for Optimal Regression Trees is therefore obtained by following the approach in Algorithm 2.1 and changing the definition of the loss function $L(\mathbb{T}, \mathbf{X}, \mathbf{y})$ in (2.26) to be the squared (or absolute) loss when the predictions in the leaves of $\mathbb{T}$ are chosen to be the mean (or median) of the points contained in that leaf according to the data $\mathbf{X}$ and $\mathbf{y}$. We can then obtain the local-search for Optimal Regression Trees with Hyperplanes by applying the modifications to Algorithm 2.1 described in Section 3.2.

Complexity analysis of local search for regression

We will now analyze the computational complexity of the local search procedures for constructing regression trees. Due to the similarity of these procedures to those for classification, we will focus here on highlighting the differences.

First, we consider ORT and ORT-H with squared loss. In order to update the error, we are required to first calculate the mean of the labels among each leaf in the tree and then calculate the squared loss of this prediction on the points of each leaf. When doing this inside BestParallelSplit and BestHyperplaneSplit, we can simply maintain the mean and variance of the labels of points in each leaf, since the squared loss of the mean prediction on a subset of points is equivalent to the variance of the labels of the subset. This means that the error update amounts to updating the mean and variance after moving a single point between branches, which has a cost of $O(1)$, compared to $O(K)$ for classification. If instead we are calculating the error from scratch, we must calculate the assignment of points to leaves for a cost of $O(nT)$ and then the mean among labels in each leaf at a cost of $O(T)$ for an overall cost of $O(nT)$, compared to $O(nT + KT)$ for OCT.

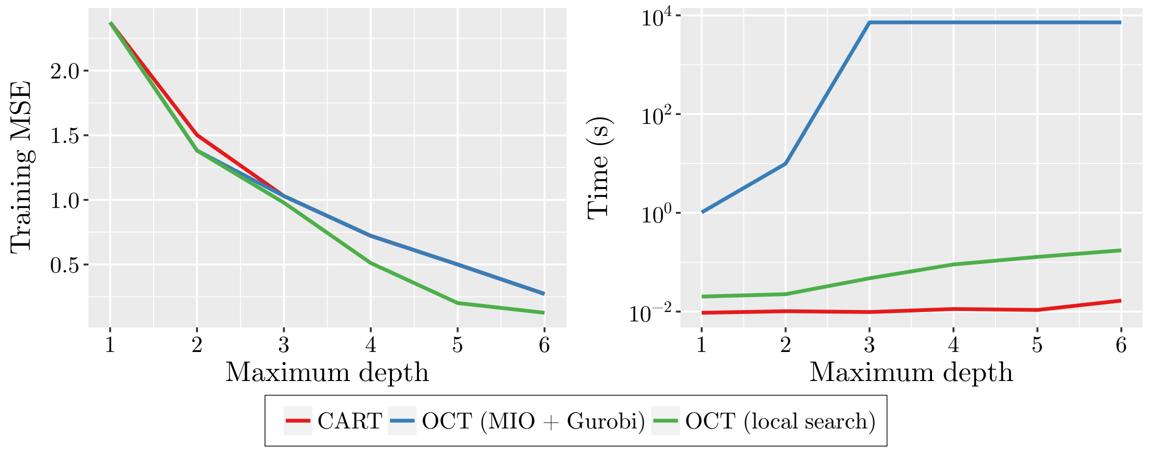
For ORT, this means that the BestParallelSplit function has a cost of $O(np)$, and therefore using the same approach as in Section 2.3 the cost of a local search iteration is $O(nT+T^2)+T\cdot O(np) = O(npT)$, compared to $O(npKT)$ for classification. If we make the same assumptions that the number of local search iterations is $O(\log n)$ and the size of the tree is also $O(\log n)$, we arrive at a total cost of $O(np\log^2 n)$. The cost of CART under similar assumptions is $O(np\log n)$, and so similar to classification, the cost of ORT is just $O(\log n)$ more than CART.

For ORT-H, the cost of BestHyperplaneSplit becomes $O(np\log n)$, which gives a cost per local search iteration of $O(npT) + T \cdot O(np\log n)$. Under the same assumptions about the number of iterations and size of the tree, the overall cost for ORT-H thus becomes $O(np\log^3 n)$, which is $O(\log n)$ more than ORT.

Next, we consider the cost of ORT and ORT-H with absolute loss. This requires us to calculate the median of the labels at each leaf followed by the absolute loss. There are approaches for updating the median and absolute loss that run in $O(\log n)$ time [115]. This means that the cost of BestParallelSplit becomes $O(np\log n)$, and the cost per local search iteration is $O(npT\log n)$. Therefore the overall cost of ORT with absolute loss under our assumptions is $O(np\log^3 n)$, an increase of $O(\log n)$ compared to the cost of ORT with squared loss. Similarly, the cost of ORT-H with absolute loss is $O(np\log^4 n)$, an increase of $O(\log n)$ over ORT-H with squared loss.

The key takeaway from this analysis is that ORT and ORT-H have the same complexity overhead with respect to CART as OCT and OCT-H when the squared loss is used. Both ORT and ORT-H incur an additional factor of $O(\log n)$ when using

Figure 4-3: Training mean squared error and running time for each method on "Hybrid Vehicle Prices" dataset.



the absolute loss, which is a significant disadvantage compared to the squared loss.

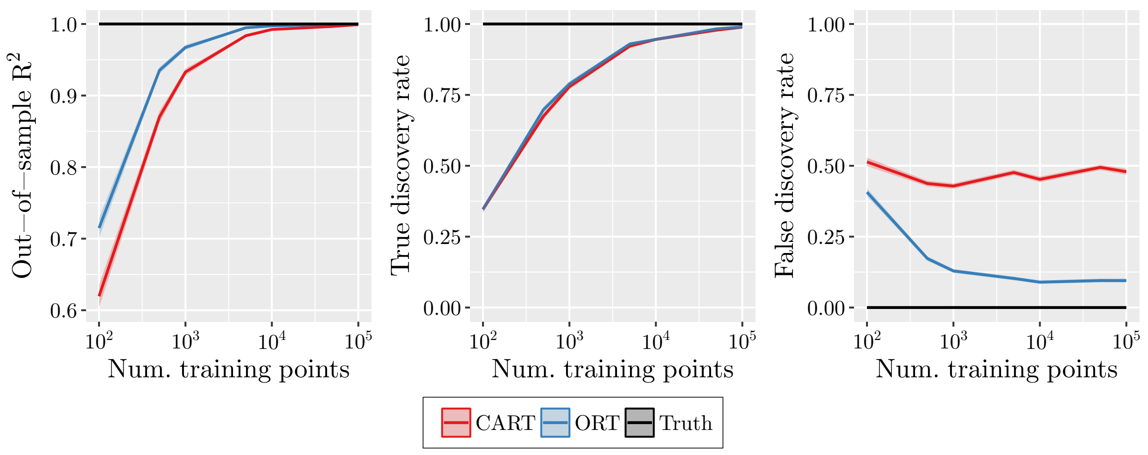
Effectiveness of the local search procedure

We return to the example of the "Hybrid Vehicle Prices" dataset to demonstrate the effectiveness of the local search procedure when applied to regression problems. Figure 4-3 compares the performance of CART and ORT using both the MIO and local search approaches. We see that ORT with local search delivers significant improvements over CART at depths 2–6 for a cost of increasing the running time about one order of magnitude. Moreover, we see at depth 2 that the local search solution coincides with the MIO solution, which we know to be provably optimal. This gives evidence that the solutions found by the local search procedure can indeed be optimal, albeit without a certificate of optimality.

4.4 Experiments with Synthetic Datasets

In this section, we evaluate the performance of Optimal Regression Trees with constant predictions, both with (ORT-H) and without (ORT) hyperplanes, on a collection of synthetic experiments. These experiments aim to provide a comparison of Optimal Regression Trees against both classical decision tree heuristics like CART and

Figure 4-4: Synthetic experiments showing the effect of training set size.



state-of-the-art methods like random forests and boosting.

The experimental setup mirrors that for classification in Sections 2.5 and 3.4 using the same default parameters unless otherwise mentioned. We randomly generate decision trees with labels drawn randomly from $U(0,1)$. We then generate training and testing datasets in the same way as for classification, generating X randomly and using the ground truth decision tree to assign the corresponding label for each point.

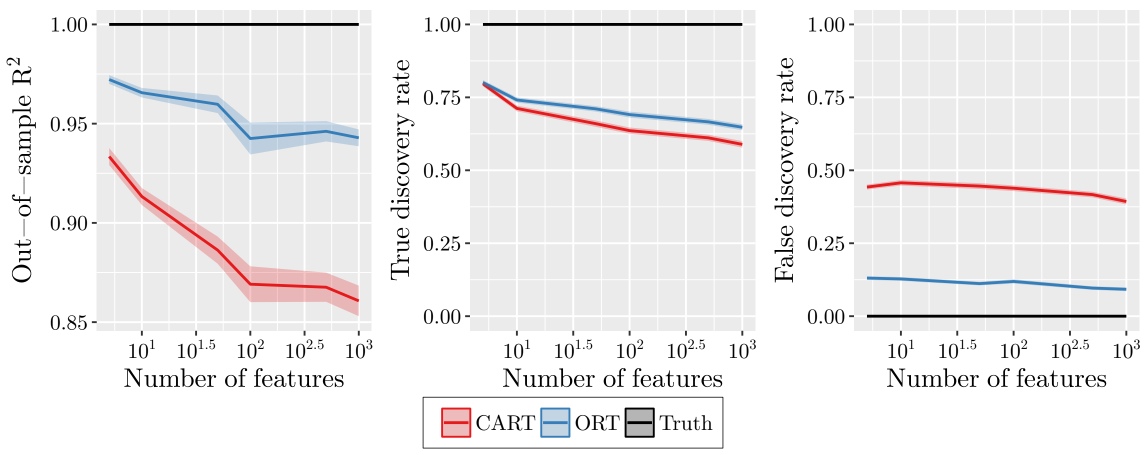
We trained ORT and ORT-H with the squared loss due to the additional computational overhead of the local search using absolute loss.

Each method was tuned to maximize R^2 using the same procedures as Sections 2.5 and 3.4. For each method, we report the out-of-sample R^2, the true discovery rate (TDR), and the false discovery rate (FDR).

CART vs. ORT

We first present a comparison between CART and OCT. Both methods solve the same optimization problem to produce a single decision tree with constant predictions in the leaves, and therefore this comparison seeks to demonstrate the impact of solving the decision tree from a global perspective, rather than greedily.

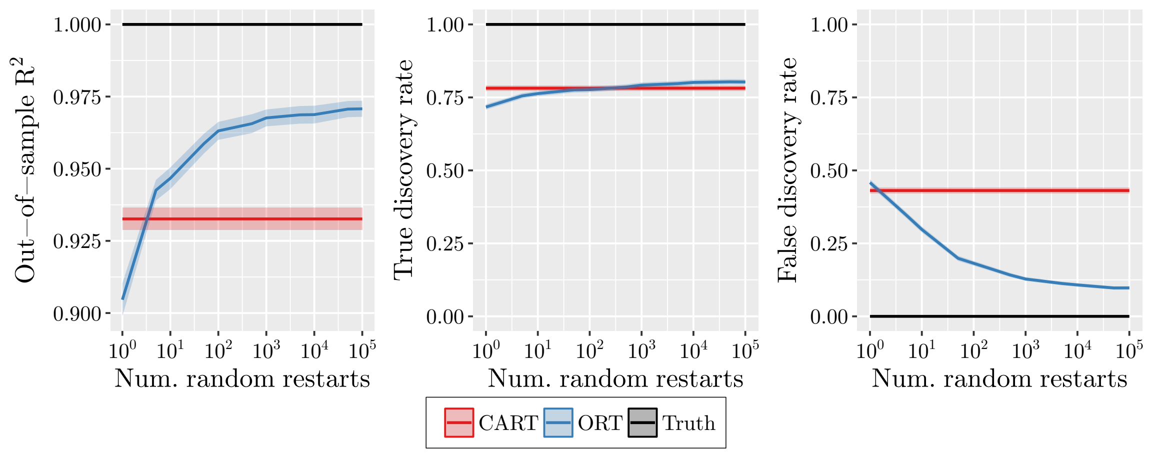
Figure 4-5: Synthetic experiments showing the effect of number of features.



The effect of the amount of training data

The first experiment shows the impact of the amount of training data relative to the complexity of the problem. The results are shown in Figure 4-4. We can see that both methods approach a perfect out-of-sample R^2 of 1 as the number of training points increases. However, ORT reaches this point with significantly fewer training points than CART, and has higher accuracy at all training sizes. The difference in R^2 is largest for smaller training sets at around 0.1, and as the training set size increases this improvement diminishes. This is clear evidence that ORT requires much less data than CART to learn the structure in the data. This result mirrors the findings of the corresponding experiment in Section 2.5, showing again that solving the decision tree problem with better optimization methods does not lead to overfitting, but rather to better solutions. The TDR is about the same for both methods, increasing as the training set size increases, as we would expect. As the training set size increases, CART's FDR remains constant around 40%, while the FDR for ORT falls to around 10% with increasing training set size. This again reinforces the results for classification, demonstrating that Optimal Trees are much better able to identify the whole truth, and nothing but the truth, in the data.

Figure 4-6: Synthetic experiments showing the effect of number of random restarts.



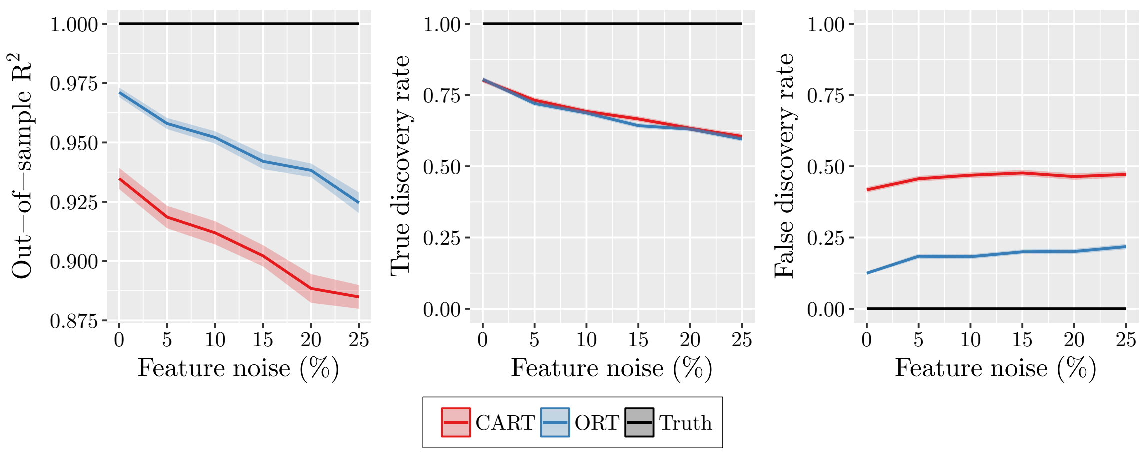
The effect of dimensionality

The second experiment investigates the effect of the dimensionality of the problem, and the results are shown in Figure 4-5. The performance of both methods decreases with increasing dimension, reflecting the increased difficulty of the problem. The R^2 falls faster with dimension for ORT than for CART, with the difference increasing to 0.08 when $p = 1000$. ORT has a small but consistent improvement in the TDR over CART. CART has an FDR around 40–45% at all dimensions, while ORT has a FDR of around 10%, which in both cases is the same as the FDR for the $n = 1000$ case in the previous experiment. This shows that the ability of ORT to learn the truth in the data and not be misled by irrelevant features is unaffected by high dimensionality.

The effect of the number of random restarts

The third experiment investigates how the number of random restarts affect the performance of ORT. The results are shown in Figure 4-6. We see that the R^2 of ORT initially increases rapidly as the number of restarts is increased, but the rate of increase begins to slow after around 100 restarts. The TDR increases steadily as the number of restarts increases, overtaking CART at around 300 restarts. The FDR initially decreases sharply, and eventually levels out at around 10%, less than a quarter of CART's 45%. Echoing the corresponding results from classification, this experiment gives strong evidence we should be using at least 100–1000 restarts to

Figure 4-7: Synthetic experiments showing the effect of feature noise.



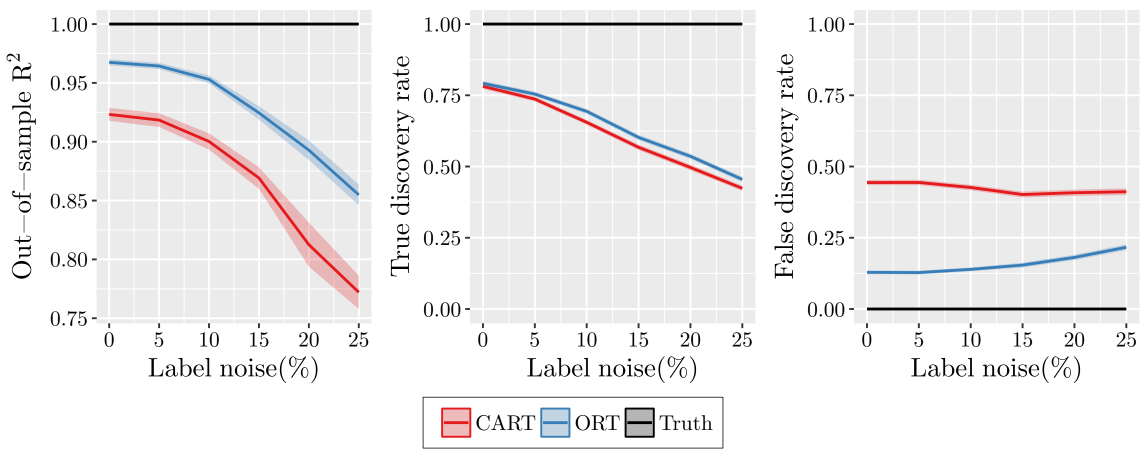
capture most of the the advantage of Optimal Trees. Further increasing the restarts beyond this point continues to provide improvements in quality, but significantly increases the computation required.

The effect of noise in the training data

The fourth experiment considers the effect of adding noise to the features of the training data. As with the previous experiments for classification, we select a random $f\%$ of the training points and add random noise $\epsilon \sim U(-0.1, 0.1)$ to every feature of these points, where f is the parameter of the experiment. Figure 4-7 shows the impact of this feature noise in the training data. As expected, the R^2 of both methods decreases as we increase the noise in the data, with ORT having an advantage of around 0.04–0.05 at all noise levels. The TDR decreases in a similar fashion to the R^2 as the noise increases, with both methods performing similarly. The FDR increases slightly with increasing noise for both methods, but ORT has an FDR less than half that of CART.

The final experiment also adds noise to the training data, but to the labels instead of the features. We introduce noise to the labels by adding i.i.d. random noise $\sim N(0, f^2)$ to each point, where f is the parameter of the experiment. Figure 2-16 shows the impact of increasing levels of such label noise in the training data. As for the previous experiment with noise, the R^2 of both methods decreases as the

Figure 4-8: Synthetic experiments showing the effect of label noise.

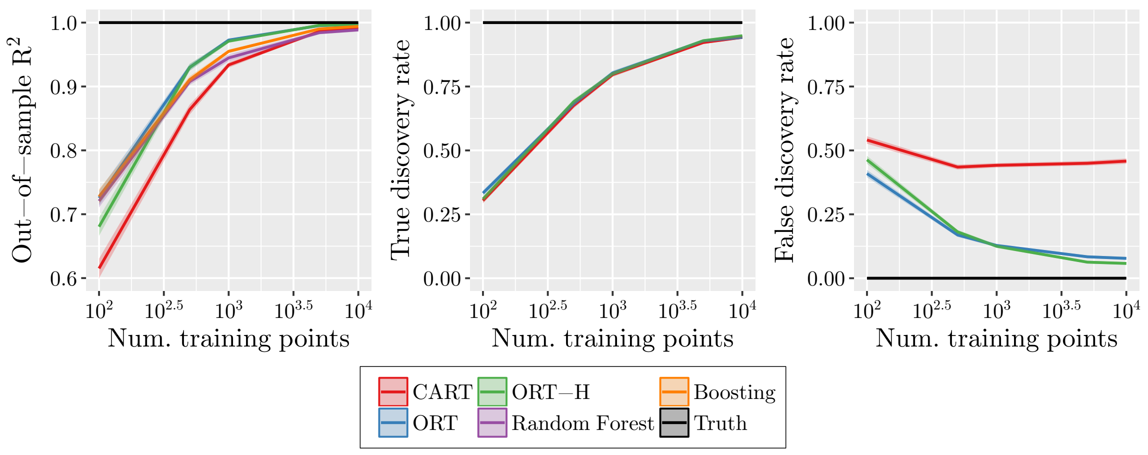


amount of noise is increased. ORT outperforms CART in terms of R^2 by around 5–7%, with no apparent dependence on the amount of noise. The TDR also decreases with increasing noise, with OCT outperforming CART by around 3–4%. The FDR of CART is largely unaffected by the label noise, while the FDR of ORT increases slightly as the amount of noise increases, growing from around 13% to around 23%, compared to CART's FDR of around 42%.

As we did in Section 2.5 for classification, we reconducted the first three experiments with the addition of both feature noise and label noise to confirm that the trends in the results presented are robust to noise. In each experiment, the results with noise are not sufficiently different to those without, other than slight decreases in performance due to the increased difficulty of the problems. This demonstrates that the advantage of ORT over CART is unchanged by the presence of noise, provided of course that the level of noise is not so high as to destroy any chance of recovering the truth in the data.

Together these experiments with noise offer strong evidence that Optimal Trees do not overfit to the noise in the data, but rather that by solving the problem optimally we are better able to filter through the noise and extract only the truth.

Figure 4-9: Synthetic experiments showing the effect of training set size.



All Methods

Next, we investigate the performance of ORT and ORT-H by comparing them to other tree-based methods for regression (CART, random forests and boosting) in order to evaluate the performance of Optimal Regression Trees against the methods used in practice.

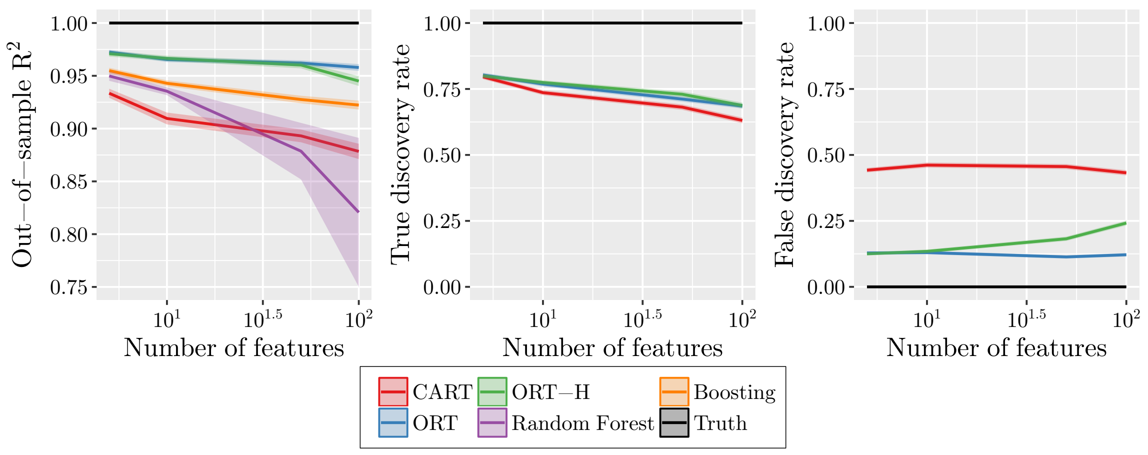
Ground truth trees with parallel splits

First, we repeat the experiments with all methods present so that we can examine the performance of ORT-H on data generated from trees with a parallel split structure, and also to compare the performance of CART, ORT and ORT-H to random forests and boosting, which have state-of-the-art accuracy. This will give us a sense of the significance of the improvements of ORT and ORT-H over CART.

The effect of the amount of training data

The first experiment measures the effect of increasing the amount of available training data. The results are shown in Figure 4-9. We see that ORT and ORT-H share the highest out-of-sample R^2 at all training set sizes other than $n = 100$, where ORT, random forests, and boosting all perform similarly and ORT-H lags slightly. All methods eventually approach perfect out-of-sample performance with enough training data, but ORT and ORT-H require much less training data to reach this limit. All

Figure 4-10: Synthetic experiments showing the effect of number of features.

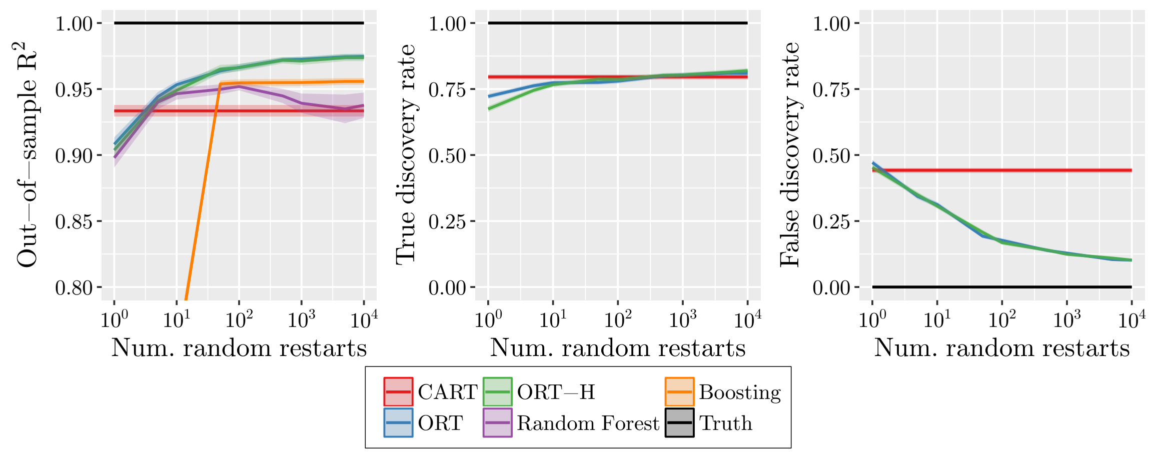


three tree methods have the same TDR, but CART has a significantly worse FDR than the optimal tree methods. The FDR of CART remains roughly constant around 50% regardless of the training set size, indicating that half of the splits in the CART model are meaningless, which causes problems when attempting to interpret the tree in a meaningful way. On the other hand, the optimal tree methods have a FDR that goes to zero as the training size increases, which coupled with the TDR approaching one indicates that these methods can successfully recover the whole truth in the data, and nothing but the truth.

The effect of dimensionality

The second experiment measures the performance of the methods while changing the number of features in the dataset, and the results are shown in Figure 4-10. We see trends that are generally similar to those for classification in Figure 3-5. The out-of-sample R^2 for all methods decreases as the number of features increases, as a consequence of the problem becoming more difficult. However, the performance of random forests falls much faster than the other methods, and this decreased performance is also accompanied by increased variance. This seems to indicate that random forests are ill-suited to regression problems with large numbers of features. ORT-H shows a drop in accuracy compared to ORT as the number of features increases, but this drop is much less significant than the corresponding drop seen for OCT-H vs

Figure 4-11: Synthetic experiments showing the effect of number of random restarts.

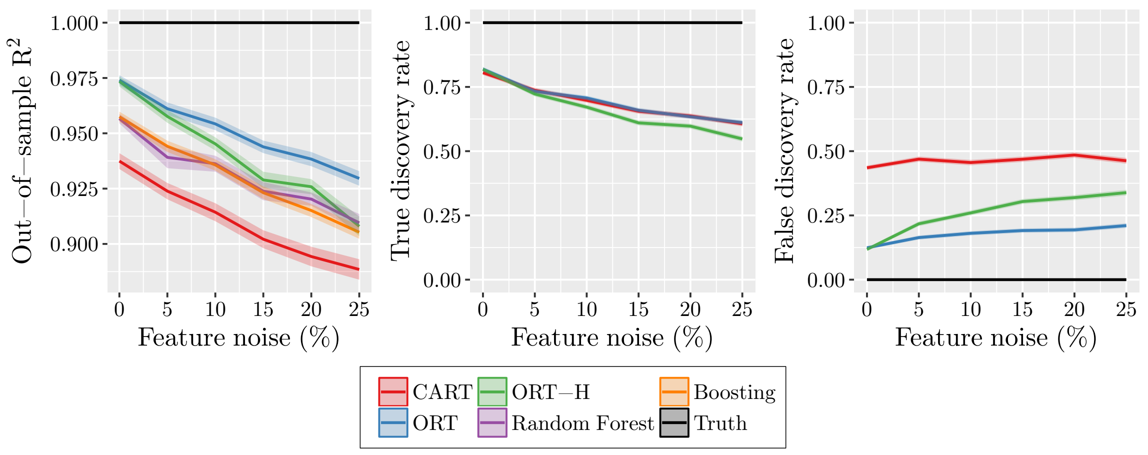


OCT. All three tree methods perform similarly in TDR. The FDR of CART and ORT remains constant with an increasing number of features, but the FDR for ORT-H begins to worsen. As with the out-of-sample R^2, this deterioration as the number of features increases is less pronounced than it was for OCT-H compared to OCT.

The effect of the number of random restarts

Next, we consider varying the number of random restarts used in the optimal tree methods. For comparison purposes, we use the number of random restarts as the number of trees in both random forests and boosting to measure how well each method performs as a function of how many trees are trained overall. The results are shown in Figure 4-11. The out-of-sample performance of ORT and ORT-H is the best, and largely levels out after 1,000 random restarts. Boosting levels out after around 100 trees, but performs significantly worse than the optimal tree methods. Random forests do not seem to be very stable, with performance that oscillates between that of CART and boosting and has much higher variance than the other methods. The TDR and FDR of ORT and ORT-H is very similar, with the TDR leveling out around 0.8, similar to CART, and the FDR decreasing towards zero, compared to around 0.45 for CART.

Figure 4-12: Synthetic experiments showing the effect of feature noise.

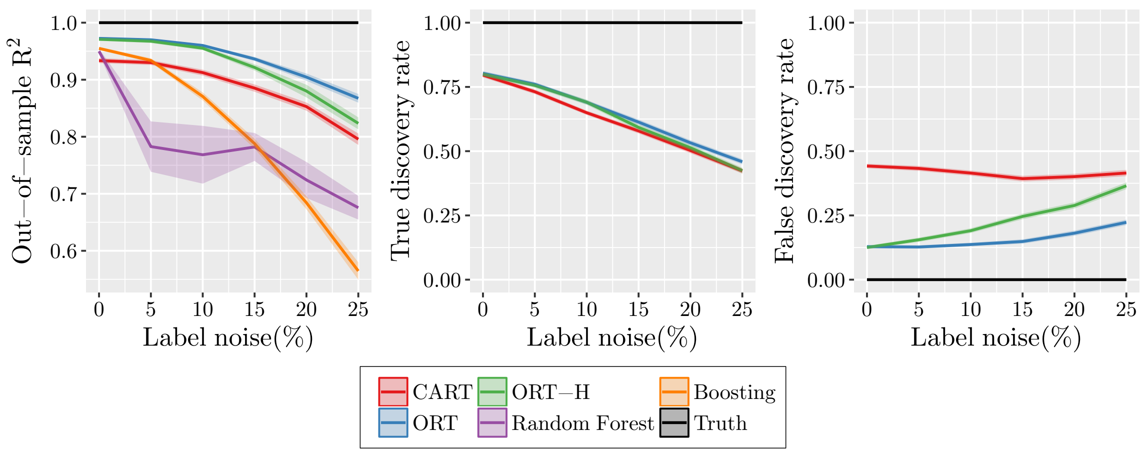


The effect of noise in the training data

The next experiment considers adding feature noise to the training data, and Figure 4-12 shows the results. We can see that the out-of-sample performance of all methods falls as the noise increases, as we would expect. ORT is the best performing method and CART the worst, with random forests and boosting performing similarly inbetween ORT and CART. ORT-H starts with performance similar to ORT, but with increasing noise falls to have performance similar to random forest and boosting, indicating it has more trouble dealing with the noise than ORT. This is mirrored in the TDR and FDR, where the TDR of ORT-H is slightly lower than the others at high levels of noise, and the FDR is increasing towards that of CART as the noise is increased. This diminished performance of ORT-H as the noise increases can be attributed to it being a more flexible and powerful model, and thus it seems to be thrown off by the noise more than the simpler ORT. That said, ORT-H is still the second-best performing method in the test, and moreover the ground truth tree is a parallel tree, so it is perhaps unsurprising that ORT has better performance when the noise is increased and the problem becomes more difficult.

We also consider adding label noise to the training data, and the results of this experiment are shown in Figure 4-13. The results for all methods largely mirror those for classification in Figure 3-9, with the exception of random forests, which seem to perform significantly worse under label noise for regression compared to

Figure 4-13: Synthetic experiments showing the effect of label noise.



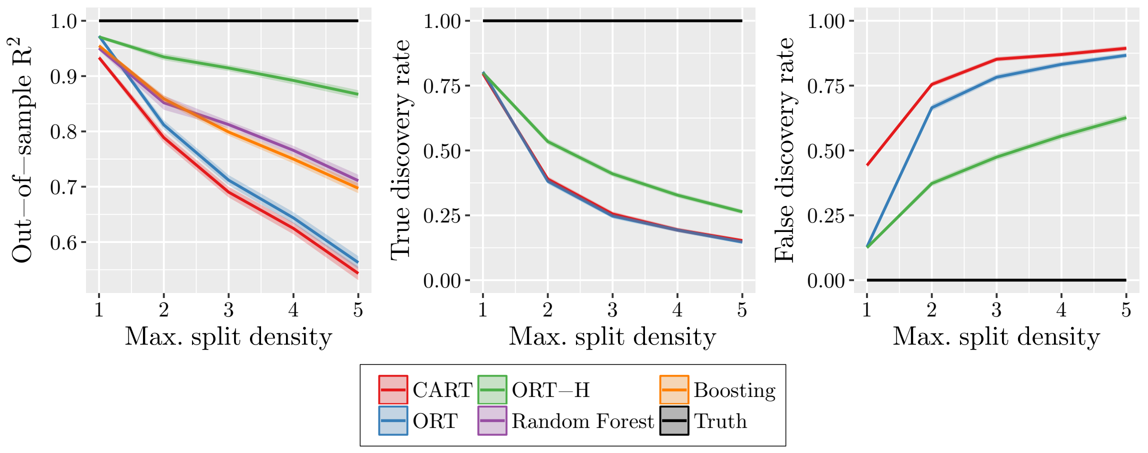
classification. The optimal tree methods and CART all decrease slightly in out-of-sample performance as the label noise increases, whereas both random forests and boosting both deteriorate significantly as the label noise is increased. At the highest level of noise, the R^2 for ORT is around 0.85 compared to around 0.7 for random forests and 0.55 for boosting, demonstrating that optimal tree methods can much better handle label noise in the training data.

Summary for ground truth trees with parallel splits

We now summarize the results of these comparisons of all methods on data generated from ground truth trees with parallel splits. In most cases, ORT and ORT-H performed similarly and were the best among all the methods. There were some cases where the performance of ORT-H was reduced compared to ORT, but even in these cases it was still the second-best performing method. We also found some cases where random forests and boosting exhibited significantly reduced performance, namely high levels of label noise, and increased numbers of features for random forests.

The key takeaway from these results is that ORT and ORT-H achieve performance at least comparable to, and often stronger than, random forests and boosting when the underlying truth in the data follows a tree structure. We therefore do not need to choose between model interpretability and performance if we have reason to believe the data has an underlying tree structure.

Figure 4-14: Synthetic experiments showing the effect of split density.



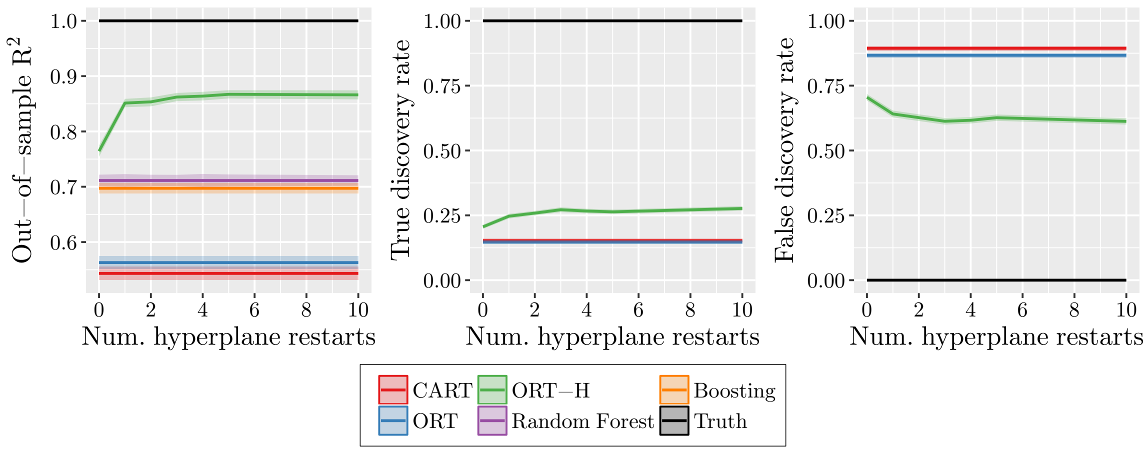
Ground truth trees with hyperplane splits

Next, we consider experiments for which the data generation process is according to a ground truth tree with hyperplane splits. These experiments aim to show the added power of ORT-H over ORT, and also to benchmark the performance of CART, random forests and boosting on these tougher problems to provide a reference for the performance of the optimal tree methods.

The effect of split density

Our first experiment with data generated from trees with hyperplane splits considers changing the maximum number of variables appearing in these hyperplane splits, which we call the split density. The results are shown in Figure 4-14, and closely mirror the corresponding results for classification in Figure 3-10. Unsurprisingly, the performance of all methods falls as the split density increases and the problem becomes harder. We see that ORT-H is the best performing method, which is again unsurprising as it can explicitly model and learn hyperplane splits. Random forests and boosting are the next best performing methods, followed by CART and ORT, indicating that aggregating trees with parallel splits leads to better performance in cases where the parallel splits do not accurately describe the structure in the data.

Figure 4-15: Synthetic experiments showing the effect of number of random hyperplane restarts.



The effect of the number of random hyperplane restarts

The second experiment with data generated by hyperplane trees aims to measure the importance of the number of random hyperplane restarts H on the performance of ORT-H. The data was generated from ground truth trees with hyperplane splits with a maximum split density of 5—the same as the number of features p. The results shown in Figure 4-15 again closely match those for the corresponding test for classification in Figure 3-11. We see that the most important change in performance occurs when we go from no hyperplane restarts to just a single restart. After that, the performance is largely level, with just small increases as H is further increased. This offers strong evidence that we should always choose $H > 0$, and to be sure that the best performance is reached, H should likely be somewhere between 5 and 10 depending on how much computational power and time is available.

Summary for ground truth trees with hyperplane splits

We saw in both experiments where the ground truth was trees with hyperplane splits that ORT-H was able to perform better than the other methods and deal with the increased complexity of the problem. We saw that ORT-H had the best performance as the density of the true hyperplane splits increased, and also that random forests and boosting outperformed CART and ORT, but were unable to reach the performance of ORT-H. This shows that aggregation of trees with parallel splits can help to model more complicated structures in the data, but is not as powerful as a model that models the structure in the data exactly. We also saw that adding just a single random hyperplane restart to the training process greatly increased the performance of ORT-H, just as it did for OCT-H.

4.5 Experiments with Real-World Datasets

In this section, we evaluate the performance of our Optimal Regression Trees against other methods across a wide sample of real-world datasets to compare their applicability in practical settings.

Experimental Setup

The setup for these experiments largely mirrors that used for classification in Sections 2.6 and 3.5. We used a diverse collection of 26 regression datasets from the UCI Machine Learning Repository [77], a data repository at the University of Porto [114], and a data repository at the University of Florida [121]. The datasets have sizes in from the hundreds to tens of thousands and a number of features typically in the tens.

As described in Section 2.6, we split each dataset three ways into training, validation and testing (50/25/25%). We tuned the hyperparameters of each method using the training and validation sets before training the final model on the combined training and validation sets with the tuned hyperparameter values. We then evaluated the R^2 of the resulting model on the test set to determine the out-of-sample performance. This procedure is repeated for five splittings of each dataset and the performances we report are averaged across these different instances.

We trained all methods with maximum depths 1–10, which in nearly every case was sufficiently deep and increasing the depth further did not improve the performance. The results for depth 10 can therefore be seen as the results when imposing no restriction on the maximum depth of each method.

Table 4.1: Full results for CART and ORT at depth 10. The best method for each dataset is indicated in bold. Positive improvements are highlighted in blue, and negative in red.

Dataset			Mean out-of-sample R^2		Mean improvement
Name	n	p	CART	ORT	
abalone	4176	7	**0.460**	0.457	-0.003 ± 0.013
ailerons	7153	40	**0.746**	0.736	-0.010 ± 0.010
airfoil-self-noise	1502	5	**0.816**	0.814	-0.002 ± 0.009
auto-mpg	391	8	**0.771**	0.771	0.000 ± 0.011
automobile	158	51	0.581	**0.590**	$+0.009 \pm 0.089$
cart-artificial	40767	10	**0.948**	**0.948**	0.000 ± 0.000
communities-and-crime	122	122	0.409	**0.507**	$+0.098 \pm 0.084$
computer-hardware	208	36	0.789	**0.858**	$+0.069 \pm 0.093$
concrete-compressive	102	7	**0.453**	0.364	-0.088 ± 0.072
concrete-flow	102	7	0.208	**0.251**	$+0.043 \pm 0.028$
concrete-slump	102	7	0.172	**0.217**	$+0.045 \pm 0.023$
cpu-act	8191	21	**0.970**	0.966	-0.004 ± 0.002
cpu-small	8191	12	**0.959**	0.959	0.000 ± 0.001
elevators	8751	18	0.686	**0.689**	$+0.003 \pm 0.003$
friedman-artificial	40767	10	0.846	**0.852**	$+0.006 \pm 0.001$
housing	505	13	0.705	**0.750**	$+0.045 \pm 0.058$
hybrid-price	152	3	**0.512**	0.464	-0.048 ± 0.057
kin8nm	8191	8	0.442	**0.498**	$+0.056 \pm 0.014$
lpga-2008	156	6	0.594	**0.624**	$+0.030 \pm 0.052$
lpga-2009	145	11	**0.802**	0.786	-0.016 ± 0.023
parkinsons-motor	5874	16	**0.159**	0.159	0.000 ± 0.021
parkinsons-total	5874	16	0.144	**0.163**	$+0.019 \pm 0.006$
vote-for-clinton	2703	9	**0.315**	0.286	-0.028 ± 0.013
wine-quality-red	1598	11	**0.286**	0.263	-0.024 ± 0.014
wine-quality-white	4897	11	0.279	**0.282**	$+0.003 \pm 0.003$
yacht-hydrodynamics	307	6	0.988	**0.991**	$+0.003 \pm 0.002$

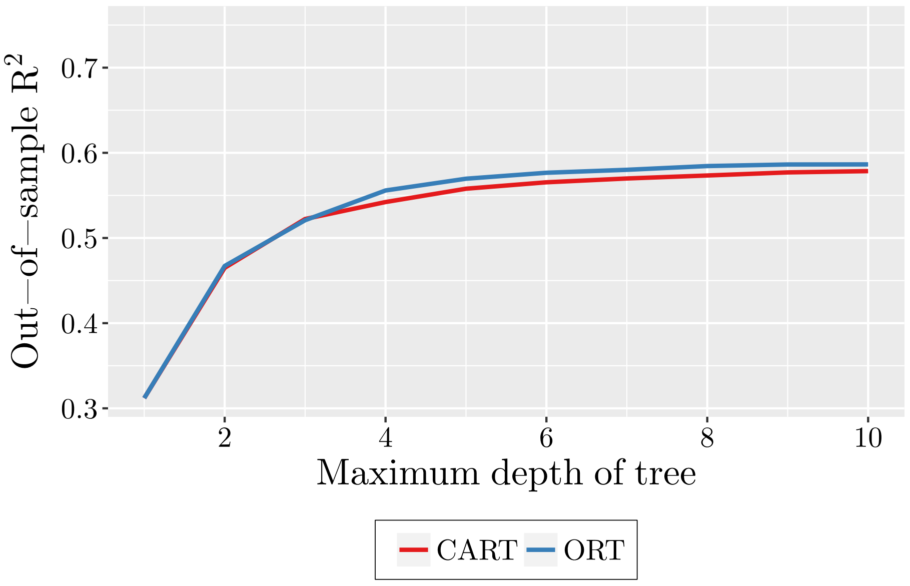
CART was trained with `minbucket` = 1 and `cp` determined using cost-complexity pruning. ORT was trained with `minbucket` = 1, $R = 100$ random restarts, and `cp` tuned using the batch pruning procedure in Algorithm 2.6. ORT-H was trained with the same parameters as ORT along with $H = 5$ random hyperplane restarts. Random forests and boosted trees were trained as described in Section 3.4.

CART vs. ORT

First, we present a comparison of the relative performances of CART and ORT. These comparisons allow us to directly measure the impact of solving the regression tree construction problem with optimization methods rather than greedily.

Table 4.1 shows the out-of-sample R^2 on each dataset for both methods with maximum depth 10, as well as the average improvement in R^2 of ORT over CART.

Figure 4-16: Mean out-of-sample R^2 for each method across all 26 datasets.



The mean out-of-sample R^2 across all datasets is shown in Figure 4-16 for both CART and ORT as a function of the maximum depth of the tree. A comparison of the accuracies is also provided in Table 4.2, showing the average performance of each method, the average improvement of ORT over CART, and the associated p-value for the statistical significance of the difference.

There is no difference in performance at depth 1 as CART is also optimal when only one split is present; unlike classification, the CART and ORT loss functions are the same for regression. The methods perform similarly at depths 2 and 3, indicating that for very shallow trees the greediness of CART does not hurt its performance. We see that ORT has a small advantage over CART of around 0.01 from depths 4–10, although this difference is not statistically significant. This indicates that ORT is able to slightly improve upon the performance of CART for deeper trees, but it appears the key factor limiting performance is the structure of the model we are fitting (a tree with parallel splits and constant predictions) rather than how well it is being optimized. Similar to classification, the ORT is able to achieve performance comparable to CART with shallower trees. For instance, the performance of CART with depth 10 (effectively no restriction) is matched by ORT with depth 6 or 7, meaning the trees are going to be more easily interpreted.

Table 4.2: Mean out-of-sample R^2 results for CART and ORT across all 26 datasets.

Maximum depth	Mean out-of-sample R^2		Mean improvement	p-value
	CART	ORT		
1	0.312	0.312	0.000	-
2	0.465	0.467	$+0.002 \pm 0.006$	0.7152
3	0.522	0.521	-0.001 ± 0.007	0.8298
4	0.542	0.556	$+0.014 \pm 0.008$	0.0820
5	0.558	0.570	$+0.012 \pm 0.008$	0.1507
6	0.565	0.577	$+0.011 \pm 0.008$	0.1681
7	0.570	0.580	$+0.010 \pm 0.008$	0.2034
8	0.573	0.584	$+0.011 \pm 0.008$	0.1607
9	0.577	0.586	$+0.009 \pm 0.008$	0.2330
10	0.579	0.586	$+0.008 \pm 0.008$	0.3086

All methods

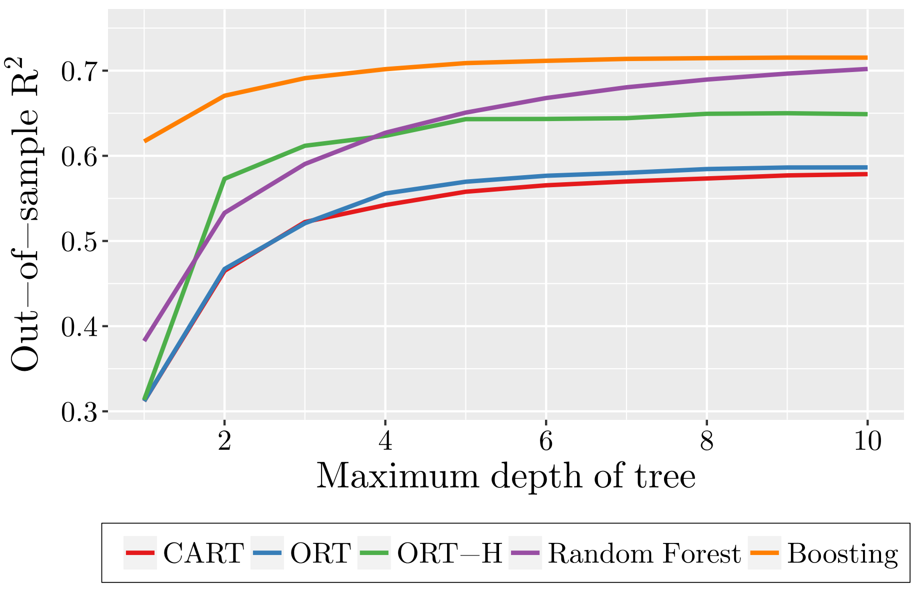
Next, we present a performance comparison of all methods considered. In addition to CART and ORT, we include results for ORT-H to investigate the impact of allowing hyperplane splits, as well as random forests and gradient boosted trees to compare our methods against those with state-of-the-art regression performance.

Figure 4-17 shows the mean out-of-sample R^2 for each method across all 26 datasets, as a function of the maximum permitted depth of the tree.

First, we compare ORT-H to the other single decision tree methods. We can see that ORT-H improves significantly upon both CART and ORT at all depths larger than 1, with an improvement in R^2 from 0.05–0.10, depending on the depth. Moreover, ORT-H with depth 2 performs comparably to CART and ORT with no depth restriction. This again demonstrates evidence for the idea introduced in Section 3.3 that a tree with hyperplane splits is not necessarily less interpretable than one with parallel splits, because the depths of the trees might be very different to achieve the same performance. Finally, we can see that the performance of ORT-H largely levels out after depth 5, indicating that we are reaching some limit on how much larger trees can help.

Next, we will compare against the two remaining methods which typically have

Figure 4-17: Mean out-of-sample R^2 for each method across all 26 datasets.



state-of-the-art performance, random forests and boosting. We can see that the strongest performer at all depths is boosting, improving upon the average R^2 of ORT-H by around 0.05. Random forests and ORT-H are comparable at depths up to 5, after which random forests continues improving and eventually reaches performance comparable to boosting at depth 10, while ORT-H levels out. We conclude that while ORT-H is able to improve significantly upon the regression trees with parallel splits, the addition of hyperplane splits is not enough to close the gap with random forests and boosting, unlike in classification.

We will now quantify these comparisons using the same procedure for multiple comparisons that was used in Section 3.5. Table 4.3 shows both the R^2 and performance rank of each method on each of the datasets, followed by the average rank of each method at the bottom of the table. We can see that boosting is the strongest method in terms of average rank, followed closely by random forests. There is then a larger gap to ORT-H, which is then followed by ORT and CART which have roughly the same average rank.

Table 4.4 shows the significance of the differences between each pair of methods, again following the procedure in Section 3.5. We can see that both boosting/random forests and ORT/CART do not have statistically significant differences in perfor-

Table 4.3: Performance results for regression methods (with maximum depth 10) on each of the 26 datasets. For each method, we show the out-of-sample R^2 followed by the method's rank on the dataset in parentheses. A rank of 1 indicates the method had the best performance on a dataset and a rank of 5 indicates the method performed worst.

Dataset	CART	ORT	ORT-H	RF	Boosting
abalone	0.460 (4)	0.457 (5)	0.519 (2)	0.477 (3)	0.536 (1)
ailerons	0.746 (4)	0.736 (5)	0.806 (3)	0.832 (2)	0.844 (1)
airfoil-self-noise	0.816 (4)	0.814 (5)	0.863 (3)	0.910 (2)	0.941 (1)
auto-mpg	0.771 (3)	0.771 (4)	0.766 (5)	0.847 (2)	0.849 (1)
automobile	0.581 (4)	0.590 (3)	0.551 (5)	0.732 (2)	0.736 (1)
cart-artificial	0.948 (1.5)	0.948 (1.5)	0.948 (4)	0.945 (5)	0.948 (3)
communities-and-crime	0.409 (4)	0.507 (3)	0.390 (5)	0.702 (1)	0.679 (2)
computer-hardware	0.789 (5)	0.858 (3)	0.797 (4)	0.929 (1)	0.929 (2)
concrete-compressive	0.453 (4)	0.364 (5)	0.569 (3)	0.690 (2)	0.792 (1)
concrete-flow	0.208 (5)	0.251 (4)	0.380 (3)	0.437 (1)	0.396 (2)
concrete-slump	0.172 (5)	0.217 (4)	0.570 (1)	0.359 (2)	0.272 (3)
cpu-act	0.970 (4)	0.966 (5)	0.975 (3)	0.981 (2)	0.985 (1)
cpu-small	0.959 (4)	0.959 (5)	0.967 (3)	0.974 (2)	0.979 (1)
elevators	0.686 (5)	0.689 (4)	0.874 (2)	0.815 (3)	0.877 (1)
friedman-artificial	0.846 (5)	0.852 (4)	0.920 (2)	0.896 (3)	0.952 (1)
housing	0.705 (5)	0.750 (4)	0.753 (3)	0.852 (2)	0.867 (1)
hybrid-price	0.512 (3)	0.464 (5)	0.509 (4)	0.615 (1)	0.577 (2)
kin8nm	0.442 (5)	0.498 (4)	0.752 (2)	0.659 (3)	0.776 (1)
lpga-2008	0.594 (5)	0.624 (4)	0.647 (3)	0.770 (2)	0.775 (1)
lpga-2009	0.802 (4)	0.786 (5)	0.848 (3)	0.890 (1)	0.885 (2)
parkinsons-motor	0.159 (4)	0.159 (5)	0.241 (3)	0.333 (2)	0.351 (1)
parkinsons-total	0.144 (5)	0.163 (4)	0.283 (3)	0.334 (2)	0.361 (1)
vote-for-clinton	0.315 (4)	0.286 (5)	0.354 (3)	0.408 (1)	0.404 (2)
wine-quality-red	0.286 (4)	0.263 (5)	0.299 (3)	0.429 (1)	0.394 (2)
wine-quality-white	0.279 (5)	0.282 (4)	0.300 (3)	0.442 (2)	0.497 (1)
yacht-hydrodynamics	0.988 (5)	0.991 (3)	0.991 (4)	0.994 (2)	0.996 (1)
Average rank	4.250	4.173	3.154	2.000	1.423

Table 4.4: Pairwise significance tests for performance of regression methods (with maximum depth 10) across 26 datasets. The comparisons are presented in order of significance. For each comparison, the strongest-performing method is both bolded and stated first in the comparison. The p-values are found using Wilcoxon signed-rank test, and the adjusted p-values are calculated using the Holm-Bonferroni method to account for multiple comparisons. All results above the dividing line are significant at the 95% level.

Comparison	p-value	Adjusted p-value
Random Forest vs. CART	$\sim 10^{-7}$	$\sim 10^{-6}$
Random Forest vs. ORT	$\sim 10^{-7}$	$\sim 10^{-6}$
Boosting vs. CART	$\sim 10^{-7}$	$\sim 10^{-6}$
Boosting vs. ORT	$\sim 10^{-7}$	$\sim 10^{-6}$
Boosting vs. ORT-H	$\sim 10^{-4}$	0.0002
ORT-H vs. CART	$\sim 10^{-4}$	0.0003
ORT-H vs. ORT	0.0008	0.0033
Random Forest vs. ORT-H	0.0051	0.0154
Boosting vs. Random Forest	0.0842	0.1684
ORT vs. CART	0.3666	0.3666

mance, but all other pairs are significantly different. This reinforces our conclusion that while ORT-H improves upon the simple trees of ORT and CART, it does not reach the state-of-the-art performance exhibited by random forests and boosting.

We conclude by summarizing these computational results with real-world regression datasets. We found that ORT likely offers a small advantage in performance over CART, but this difference is not statistically significant. ORT-H improved significantly over both CART and ORT, showing that allowing for hyperplane splits leads to greatly increased performance. However, ORT-H was still significantly weaker than random forests and boosting for regression problems, unlike in classification where the addition of hyperplane splits led to OCT-H having comparable performance with both boosting and random forests.

4.6 Conclusions

In this chapter, we applied the approach we developed for Optimal Classification Trees to the problem of regression. Following the approach of CART, we restricted our attention to regression trees that simply make a constant prediction in each leaf of the tree.

We formulated the task of finding the Optimal Regression Tree using MIO, which required only slight modifications to the MIO formulation for Optimal Classification Trees. Empirically, we found that this MIO formulation was not practically solvable even on very small datasets, and so we adapted the same local search procedure that is used to train Optimal Classification Trees to also construct regression trees. This local search method runs in times much faster than the MIO formulation, and is able to deliver substantial improvements in objective value over the solutions found by the MIO solver with a generous time limit.

We again conducted extensive computational experiments to validate and benchmark the Optimal Regression Trees against existing methods. Our experiments with synthetic data largely mirrored the results of the corresponding synthetic experiments for classification; our Optimal Tree methods deliver significant improvements over CART in terms of out-of-sample R^2, and also construct trees with significantly fewer false positive splits, meaning that they can be more safely interpreted. The experiments with real world 26 datasets showed that ORT improved slightly upon CART, with ORT-H further improving significantly upon both. However, unlike in classification, the addition of hyperplane splits did not close the gap with the state-of-the-art methods, which delivered higher performance than ORT-H.

Overall, the results of our experiments show that while our optimization approach and the addition of hyperplane splits add value to the regression tree problem, we are not able to reach state-of-the-art performance with these alone. This is likely due to our restriction that we make only constant predictions in the leaves, and so in the next chapter we will consider using more sophisticated prediction functions in the leaves.

Chapter 5

Optimal Regression Trees with Linear Predictions

CART and other classical methods for regression trees generate trees that make only constant predictions in each leaf of the tree. The predictions made by such trees correspond to a piecewise constant model over the training space, and as such these trees may have trouble accurately capturing the structure in the data without fitting very deep trees and sacrificing some interpretability.

The key reason that constant predictions are used in place of more sophisticated predictive models is computational speed. When finding the best split at a node, a predictive model must be fit in each leaf node when evaluating each candidate split. For a normal split, this means we have to fit $O(np)$ regression models in order to find the best split, which can be prohibitively expensive if fitting the predictive model is non-trivial.

Most attempts in the literature to fit more comprehensive regression models in the leaves artificially limit the power of the regression tree to increase the computational tractability of their approach. The existing methods are also all greedy in nature, bringing with them all the inherent disadvantages we have demonstrated thus far.

The simplest approach is to first train a regression tree with constant predictions and then apply the more sophisticated models in the leaves, thus avoiding the cost of repeatedly fitting models during training. The main drawback to such approaches is the disconnect between the tree structure and prediction functions; the tree tends to be larger than needed due to the limited power of constant predictions, and the splits are not chosen with the prediction functions in mind. M5 [105] uses a similar approach to this, where a tree with constant predictions is first trained. At each node in the tree, a linear regression model is fit, using as predictors only the variables that appear in splits below this node. If this new linear models outperforms the constant predictions of the subtree rooted at this node, the subtree is deleted and replaced with the linear model.

Another approach taken by many methods [34, 81, 63] is to separate the task of choosing the split variable from the task of choosing the split threshold. These methods use hypothesis testing to identify the split variable that is "most significant" in the model, and then proceed to find the optimal split on this variable. This means that only $O(n)$ predictive models need to be fit to decide upon a split. However, this speedup clearly comes at the cost of potentially making the wrong decision regarding which variable to split on.

It is also possible to reduce the computation required by limiting the power of the models fit in each leaf. Kim et al. [72] propose a method where each leaf model is limited to two variables, (chosen by stepwise regression) which also has the benefit of improving the interpretability of the model. The main drawback is that the algorithm has no ability to fit more complicated regression models when advantageous to do so; it is always restricted in power.

Ideally, the regression models in each leaf should be robust to fluctuations in the data. This would ensure that they do not change too much when testing candidate splits, thus allowing us to zero in on the correct split location. Additionally, the models in the leaves should be only as complicated as necessary to fit the data in the leaf, and no more; if the data is well-described by few or no variables, this is the model that should be used, but if more complexity is needed, we should use a more detailed model. We desire simple models for interpretability and to avoid overfitting, but do not want to blindly impose these conditions and sacrifice performance. Finally, the algorithm needs to be designed in a fashion that permits efficient training of such models without requiring us to limit the power of the tree.

In this chapter, we detail extending the Optimal Trees framework to generate Optimal Regression Trees with linear predictions in the leaves. We demonstrate that the training process can be conducted efficiently and that the resulting trees have state-of-the-art performance whilst maintaining interpretability.

5.1 ORT with Linear Predictions via MIO

In this section, we will modify the MIO formulation for Optimal Regression Trees in Section 4.2 to generate trees with linear predictions rather than constant predictions.

In the previous MIO formulation, we used the variables β_t to represent the constant predictions made in each leaf, and used these predictions to set the fitted values f_i with the constraint (4.1). It is straightforward to extend this approach to use linear predictions instead. We will use the variables $\boldsymbol{\beta}_t$ and β_{0t} to track the prediction at each leaf, which takes the form

$$y(\mathbf{x}) = \boldsymbol{\beta}_t^\mathsf{T}\mathbf{x} + \beta_{0t}$$

We then update the constraint (4.1) with the new prediction function:

$$f_i = \sum_{t \in \mathcal{T}_L} (\boldsymbol{\beta}_t^\mathsf{T}\mathbf{x}_i + \beta_{0t}) z_{it}, \quad \forall i \in [n], \tag{5.1}$$

which linearizes in the same fashion to give

$$-M_f(1 - z_{ik}) \leq f_i - (\boldsymbol{\beta}_t^\mathsf{T}\mathbf{x}_i + \beta_{0t}) \leq M_f(1 - z_{ik}), \quad \forall i \in [n],\ t \in \mathcal{T}_L, \tag{5.2}$$

where again M_f is a sufficiently large constant. We can obtain a set of bounds for the value of M_f by noting that the objective value of any feasible solution to the problem, denoted z_F, provides an upper bound on the loss at any single point, by considering a scenario in which all the loss is placed on a single point and because the optimal loss must be no larger than the loss of any feasible solution. For squared loss, this leads to the inequality:

$$(f_i - y_i)^2 \leq z_F,\ \forall i \in [n] \implies |f_i| \leq \sqrt{z_F} + \max_i |y_i|$$

This implies that at optimality, f_i and $(\boldsymbol{\beta}_t^\mathsf{T}\mathbf{x}_i + \beta_{0t})$ can be at most $2(\sqrt{z_F} + \max_i |y_i|)$ apart, and so this is a valid value for M_f. We can apply similar reasoning to the absolute loss case to reach the value $M_f = 2(z_F + \max_i |y_i|)$. In both cases, we use the best feasible solution value we have available to us, of which there is always at least one, being the baseline solution.

We also want the ability to restrict the regression predictions in each leaf to avoid overfitting and to assist with interpretability. We do this by penalizing the norm of the regression coefficient vector in each leaf, leading to the following objective:

$$\frac{1}{\hat{L}} \sum_{i=1}^{n} L_i + \alpha \cdot C + \lambda \sum_{t \in \mathcal{T}_L} \|\boldsymbol{\beta}_t\|, \tag{5.3}$$

where the parameter λ controls the degree of regularization. We might choose the L-1 norm to control the magnitude of the coefficients and lead to more robust solutions, so that they will not vary much as we make small changes inside the tree. In this case, the norm becomes a sum of absolute values which can be linearized with standard linear optimization techniques. Alternatively, we can use the L-0 norm to restrict the number of non-zero coefficients in each regression prediction to directly pursue interpretability. In this case, we use the binary variables r_{jt} to track whether the jth feature is used in the tth regression equation:

$$-M_r r_{jt} \leq \beta_{jt} \leq M_r r_{jt}, \quad \forall j \in [p],\ t \in \mathcal{T}_L \tag{5.4}$$

where M_r is a sufficiently large constant. We do not have an explicit method of calculating a small, feasible value for this constant. It must be at least $\max_{j,t} |\beta^*_{jt}|$ where β^* is the optimal solution. However, note the data can be normalized and so we expect the optimal solution to not be too large, and therefore a relatively small constant should suffice in most cases. We then use the r_{jt} variables to calculate the value of the norm in the objective:

$$\frac{1}{\hat{L}}\sum_{i=1}^{n} L_i + \alpha \cdot C + \lambda \sum_{t \in \mathcal{T}_L} \sum_{j=1}^{p} r_{jt}, \tag{5.5}$$

We obtain the complete formulation for Optimal Regression Trees with linear predictions by using these new variables and constraints in place of the corresponding constant counterparts in the formulations from Section 4.2. For instance, the following is the ORT formulation (4.2) with linear predictions and using L-0 regularization:

$$\begin{aligned}
\min \quad & \frac{1}{\hat{L}}\sum_{i=1}^{n} L_i + \alpha \cdot C + \lambda \sum_{t \in \mathcal{T}_L} \sum_{j=1}^{p} r_{jt} && \qquad (5.6)\\
\text{s.t.} \quad & L_i \geq (f_i - y_i)^2, && \forall i \in [n],\\
& f_i - (\boldsymbol{\beta}_t^{\mathsf{T}}\mathbf{x}_i + \beta_{0t}) \geq -M_f(1 - z_{ik}), && \forall i \in [n],\ t \in \mathcal{T}_L,\\
& f_i - (\boldsymbol{\beta}_t^{\mathsf{T}}\mathbf{x}_i + \beta_{0t}) \leq +M_f(1 - z_{ik}), && \forall i \in [n],\ t \in \mathcal{T}_L,\\
& -M_r r_{jt} \leq \beta_{jt} \leq M_r r_{jt}, && \forall j \in [p],\ \forall t \in \mathcal{T}_L\\
& C = \sum_{t \in \mathcal{T}_B} d_t, &&\\
& \mathbf{a}_m^{\mathsf{T}}\mathbf{x}_i \geq b_m - (1 - z_{it}), && \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{R}(t),\\
& \mathbf{a}_m^{\mathsf{T}}(\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), && \forall i \in [n],\ t \in \mathcal{T}_L,\ m \in \mathcal{L}(t),\\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, && \forall i \in [n],\\
& z_{it} \leq l_t, && \forall t \in \mathcal{T}_L,\\
& \sum_{i=1}^{n} z_{it} \geq N_{\min} l_t, && \forall t \in \mathcal{T}_L,\\
& \sum_{j=1}^{p} a_{jt} = d_t, && \forall t \in \mathcal{T}_B,\\
& 0 \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B,\\
& d_t \leq d_{p(t)}, && \forall t \in \mathcal{T}_B \setminus \{1\},\\
& z_{it}, l_t \in \{0,1\}, && \forall i \in [n],\ k \in [K],\ t \in \mathcal{T}_L,\\
& r_{jt} \in \{0,1\}, && \forall j \in [p],\ t \in \mathcal{T}_L,
\end{aligned}$$

$$a_{jt}, d_t \in \{0,1\}, \qquad \forall j \in [p],\ t \in \mathcal{T}_B.$$

This MIO formulation shares the complexity of the corresponding constant prediction counterpart, so the formulations with squared loss are quadratic MIO problems and those with absolute loss are linear MIO problems. The addition of the binary variables r_{jt} and more big-M constraints make these models harder to solve than their constant prediction counterparts. These formulations exhibit the same issues with scaling as the other MIO formulations due to the number of variables required as the training set size increases.

5.2 ORT with Linear Predictions via Local Search

In this section, we will adapt the same local search procedure as before to generate regression trees with linear predictions in an efficient manner.

As mentioned previously, one of the main challenges in efficiently constructing a regression tree with linear predictions is the increased cost of fitting regression models at each stage of the search process rather than simply predicting the mean, which can be done in constant time. Classical methods for solving least-squares regression problems using QR-decompositions or bidiagonalization require $O(np^2)$ time [23]. Moreover, there is limited ability to efficiently update the solution after small changes during the search procedure, and the solutions produced by this approach are dense and highly sensitive to noise in the data, meaning that the solution might change significantly after the addition or removal of a single point in a leaf. All of these are undesirable traits to embed inside the loss function of our local search.

Instead, our approach makes use of the GLMNet algorithm [48], which solves the standard elastic net regression problem:

$$\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \beta_0 - \mathbf{x}_i^\mathsf{T} \boldsymbol{\beta}\right)^2 + \lambda P_\alpha(\boldsymbol{\beta}) \qquad (5.7)$$

where

$$P_\alpha(\boldsymbol{\beta}) = (1-\alpha)\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \alpha\|\boldsymbol{\beta}\|_1 \tag{5.8}$$

If we set $\alpha = 1$ to only use L-1 regularization (yielding the LASSO [111]), and denote the fitted values with $f_i = \beta_0 + \mathbf{x}_i^\mathsf{T}\boldsymbol{\beta}$, we can rewrite the problem in a simpler form as

$$\min_{\beta_0,\boldsymbol{\beta}} \frac{1}{2n}\sum_{i=1}^n (y_i - f_i)^2 + \lambda\|\boldsymbol{\beta}\|_1 \tag{5.9}$$

The GLMNet algorithm solves this regression problem very efficiently using coordinate-descent. The method iteratively updates each entry in the solution vector at very low cost by exploiting the sparsity of the solution vector; if there are k non-zeros in the solution, the cost of a single coordinate-descent cycle is $O(kp)$ plus $O(np)$ any time a zero entry of the solution vector becomes non-zero (which is rare due to the inherent robustness of the method). Compared to the least-squares cost of $O(np^2)$, the cost of this coordinate descent approach at $O(kp)$ plus the occasional $O(np)$ represents a significant speedup, especially when the solution vector is very sparse ($k \ll p$).

In addition to the fast updates, a coordinate-descent approach also permits warmstarting the solution process using an existing solution. This is crucial when combined with the inherent robustness of the solutions to this problem, as the solutions change very little at each stage of the search process, so it typically takes very few iterations to reach the optimal solution when warmstarting with the optimal solution from the previous step.

Now we will see how we can apply the coordinate-descent approach of GLMNet to Problem (5.6) with L-1 regularization. Our overall objective is of the form

$$\min \frac{1}{\hat{L}}\sum_{i=1}^n (y_i - f_i)^2 + \lambda\sum_{t\in\mathcal{T}_L} \|\boldsymbol{\beta}_t\|_1, \tag{5.10}$$

that is, we want to minimize the squared loss of the fitted values f_i with a regularization penalty on the regression coefficients $\boldsymbol{\beta}$. Let $\mathcal{I}_t$ denote the set of points contained in leaf t. We can then rewrite the objective as

$$\min \sum_{t \in \mathcal{T}_L} \left(\frac{1}{\hat{L}} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \lambda \|\boldsymbol{\beta}_t\|_1 \right). \tag{5.11}$$

Note that this objective is separable in t, and so we can solve for each leaf separately. The problem faced in each leaf is

$$\min \frac{1}{\hat{L}} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \lambda \|\boldsymbol{\beta}_t\|_1, \tag{5.12}$$

which we seek to solve using the GLMNet algorithm. We can do so by rewriting this problem into the same form as (5.9):

$$\min \frac{1}{2\,|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \frac{\lambda \hat{L}}{2\,|\mathcal{I}_t|} \|\boldsymbol{\beta}_t\|_1, \tag{5.13}$$

and so therefore we can solve the regression problem at each leaf t using the GLMNet algorithm with $\lambda_{GLM} = \lambda \hat{L} / 2\,|\mathcal{I}_t|$ on the subset of data contained in that leaf. Doing this in each leaf separately will yield regression coefficients that minimize the overall objective (5.10).

The local search therefore proceeds as for constant-prediction regression trees in Section 4.3, except that each time we are required to evaluation the loss of the tree, we update the current regression predictions in each leaf by performing coordinate-descent cycles for the problems (5.13) until converged. We then sum the final objective values of these problems across the leaves (with each leaf weighted by $2\,|\mathcal{I}_t| / \hat{L}$) to get the total loss of the tree.

Hyperparameter tuning

Using linear regression predictions in each leaf introduces the regularization parameter λ as a new tuning parameter that needs to by specified. It would be possible to use the same procedure as in Algorithm 2.7 and add an additional outer loop over possible values of λ, but this would lead to an expensive two-dimensional grid search.

Instead, we have found the following procedure to give good results empirically, based on our observation that the effects of depth and λ on the solution quality are typically independent. First, the depth is tuned using Algorithm 2.7. We then fix this depth, and tune λ using Algorithm 2.7 with the loop over depth replaced by a loop over λ. We take the best value of λ and the corresponding value of α as the final tuned hyperparameter values.

5.3 Experiments with Synthetic Datasets

In this section, we use experiments with synthetic datasets to examine the performance of Optimal Regression Trees with linear predictions. There are two main goals of these experiments. The first is to expose the trees with linear predictions to data where trees with constant predictions are optimal, and determine whether the trees with linear predictions overfit the data. The second goal is to evaluate the performance of trees with linear predictions on more complicated datasets where the trees with constant predictions do not perform strongly, and see whether the more powerful prediction functions in each leaf lead to better performance on such datasets.

The experimental setup is identical to that in Section 4.4, with the addition of the Optimal Regression Trees with linear predictions, both with (ORT-HL) and without (ORT-L) hyperplane splits. We used Algorithm 2.7 to tune ORT-L and ORT-LH, with outer loops over both the maximum depth and regularization parameter λ.

Ground truth trees with constant predictions

Our first aim is to compare the performance of the methods on datasets that have been generated according to a ground truth tree with constant predictions. This will allow us to determine whether permitting more powerful predictive models in the leaves causes the model to overfit the data.

Figure 5-1: Synthetic experiments showing the effect of training set size.

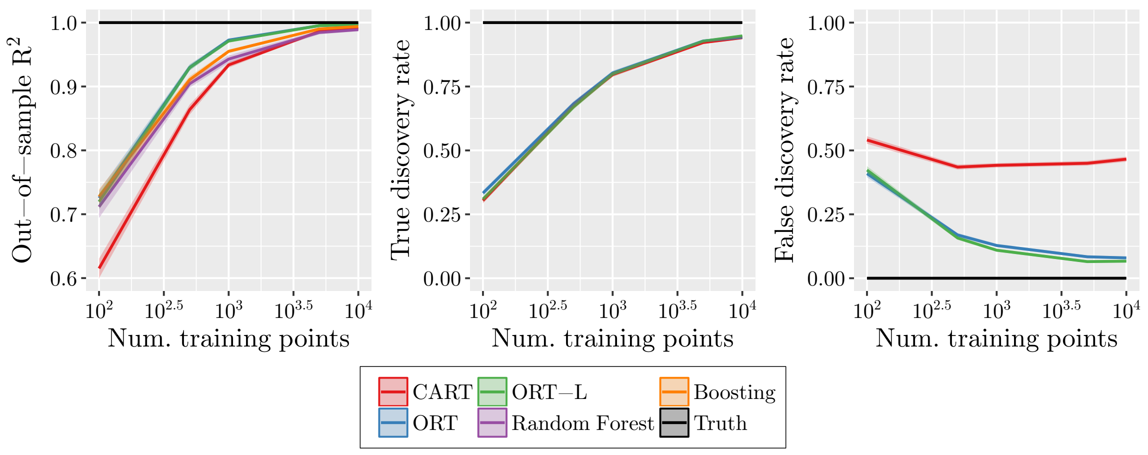
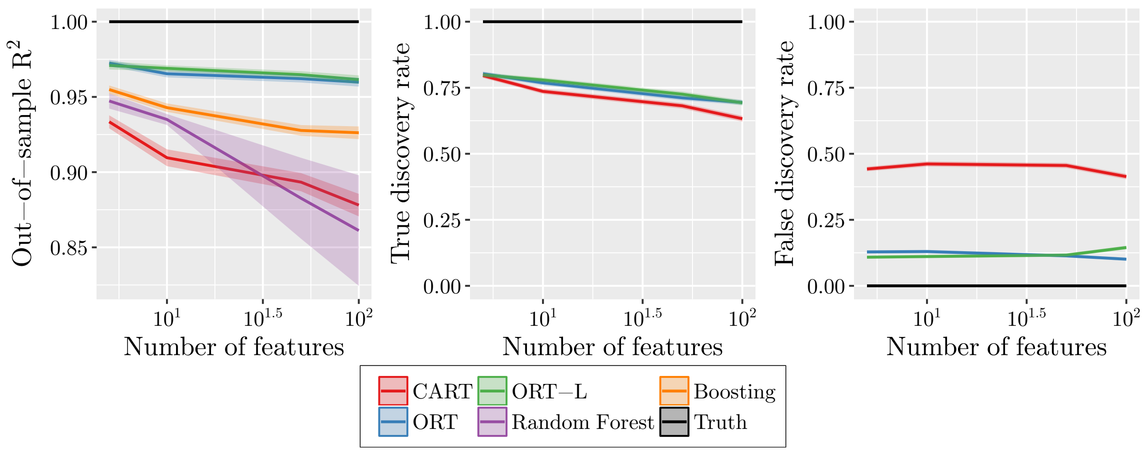


Figure 5-2: Synthetic experiments showing the effect of number of features.



The effect of the amount of training data

In the first experiment we measure the impact of increasing amounts of training data on the performance of each method. The results are shown in Figure 5-1. We can see that in all three metrics, ORT and ORT-L perform near-identically, and are both the strongest methods in terms of out-of-sample R^2.

The effect of dimensionality

The second experiment measures the effect of the number of features in the dataset on the performance of the methods. Figure 5-2 shows the results. Again we see that the performance of ORT and ORT-L is similar, with ORT-L actually performing slightly

Figure 5-3: Synthetic experiments showing the effect of number of random restarts.

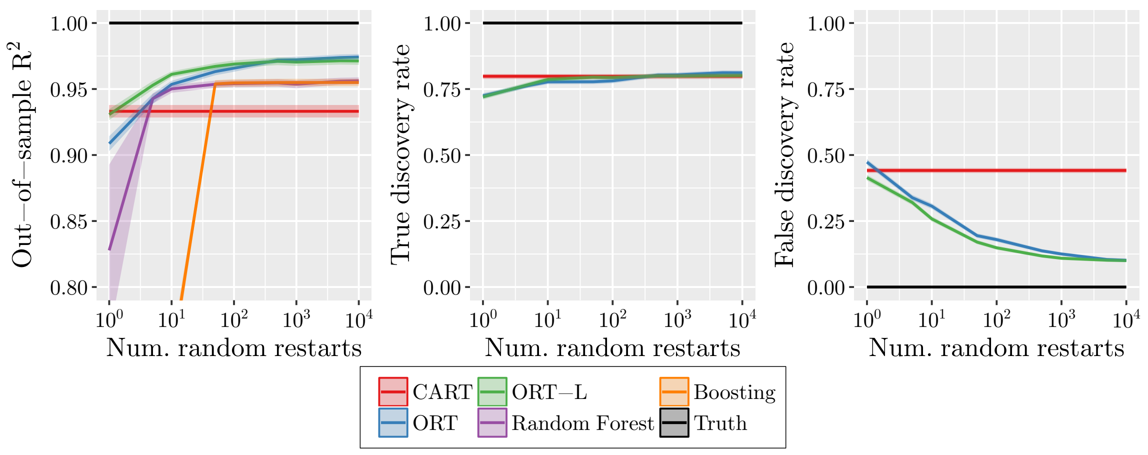
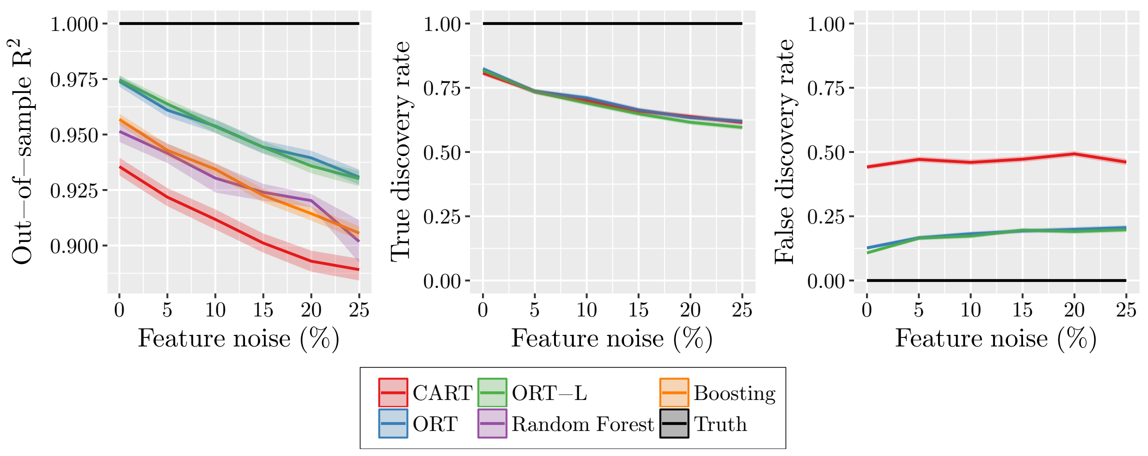


Figure 5-4: Synthetic experiments showing the effect of feature noise.

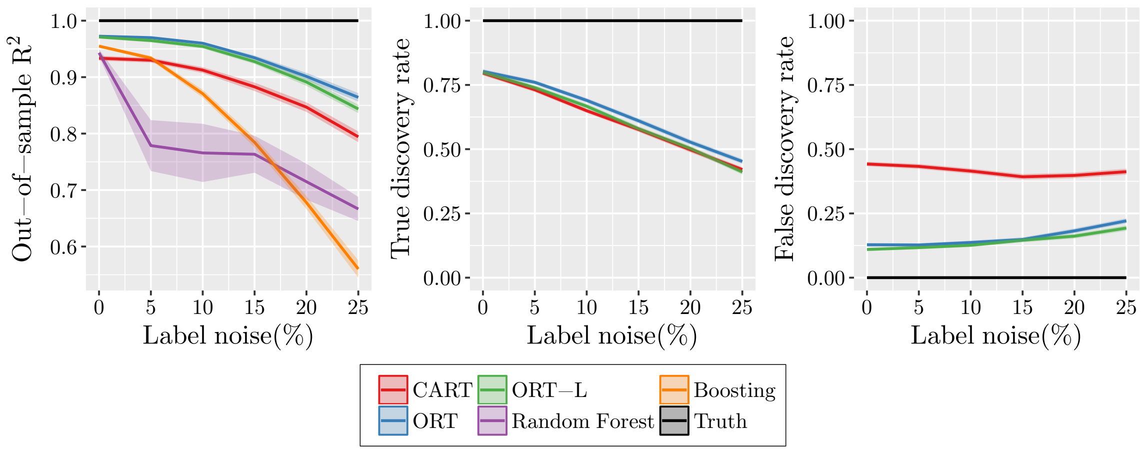


stronger than ORT in places.

The effect of the number of random restarts

The next experiment considers varying the number of random restarts used to train the optimal tree methods (and the number of trees used in the ensemble methods). The results are presented in Figure 5-3. ORT and ORT-L again perform similarly, although ORT-L has a sizable advantage in R^2 for very small numbers of random restarts, as well as a slightly lower FDR over all values.

Figure 5-5: Synthetic experiments showing the effect of label noise.



The effect of noise in the training data

The final two experiments show the effect of noise in the training set on the performance of the methods. Figures 5-4 and 5-5 show the results for each method under the addition of increasing noise in the features and labels, respectively. We see in both experiments that the performance of ORT and ORT-L is roughly similar.

Summary for ground truth trees with constant predictions

In all of the experiments using data generated by ground truth trees with constant predictions, we observed that the performance of ORT and ORT-L was near-identical. This gives us strong evidence that despite the increased power of ORT-L, it is not at risk of overfitting data that are described by simpler models. In particular, our validation procedure is able to correctly identify both the correctly-sized tree but also the correct degree of regularization in the linear predictions for a given dataset.

Ground truth trees with linear predictions

Next, we consider generating the data from ground truth trees with linear prediction models in the leaves. This will allow us to measure the ability of Optimal Trees with linear predictions to fit more complicated datasets where Optimal Trees with constant predictions perform poorly.

Figure 5-6: Synthetic experiments showing the effect of regression density.

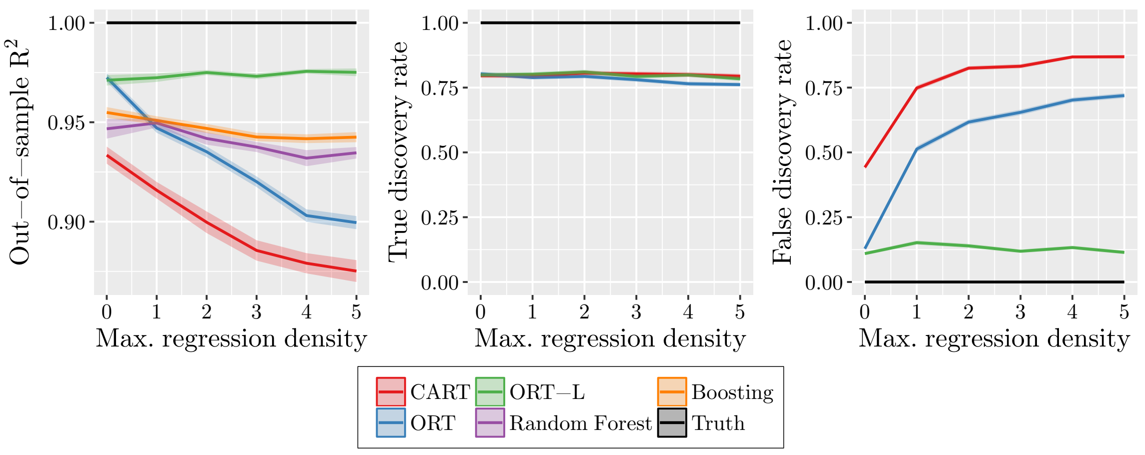
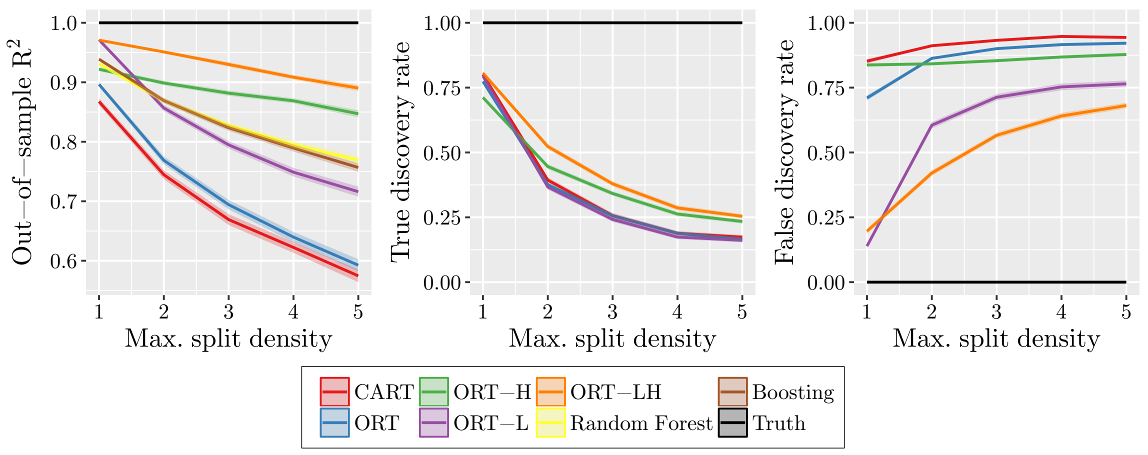


Figure 5-6 shows an experiment where we adjust the maximum density of the regression equations in each leaf of the ground truth tree. In each leaf of the ground truth tree, we generate a random linear prediction model with a maximum number of coefficients given by the *regression density*, and then use these models to generate the labels for the data. In terms of R^2, we see that ORT-L performs consistently regardless of the regression density, indicating that we are able to correctly identify the correct tree and also the correct amount of regularization in the linear regression models in the leaves. On the other hand, CART and ORT both drop off in performance as the regression density increases and their constant predictions become increasingly poor at modeling the truth in the data. Random forests and boosting also decrease in performance as the regression density increases, but this decrease is a lot slower than for CART and ORT, indicating that these methods are better able to approximate the linear relationships in the data. All three tree methods perform similarly in terms of TDR, but as we might expect the FDR of the parallel tree methods increases sharply with the regression density, as more splits are required to model the linear trends in the data, and such splits are not related to the generating ground truth. Conversely, the FDR of ORT-L is constant as the regression density is increased, showing that it can learn the whole truth and nothing but the truth regardless of the complexity of the true linear regression models that generated the data.

Figure 5-7: Synthetic experiments showing the effect of split density.



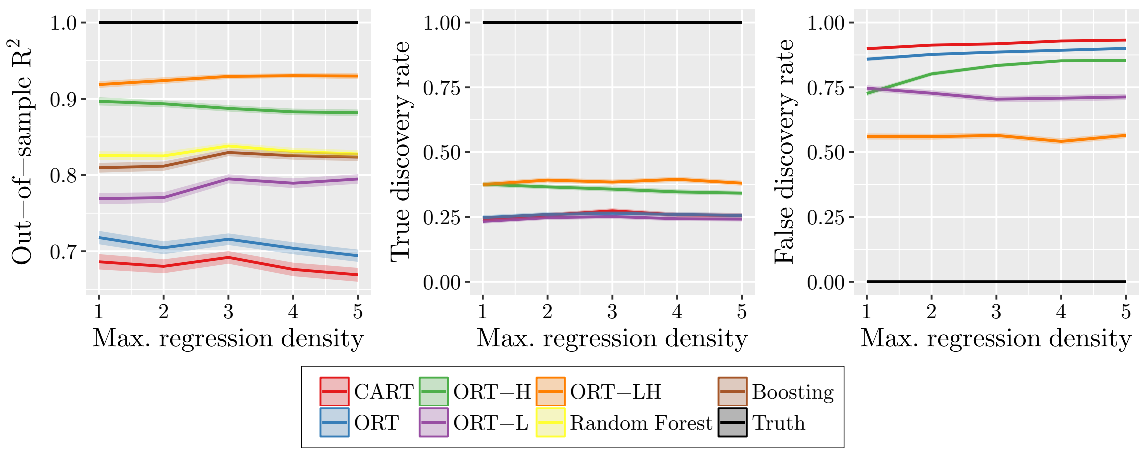
Ground truth trees with linear predictions and hyperplane splits

Finally, we consider generating ground truth trees that have both linear predictions in the leaves, and hyperplane splits at the nodes. This allows us to benchmark the performance of ORT-LH and its ability to learn complicated structure in the data.

First, we consider varying the maximum density of the hyperplane splits in the ground truth tree, with the maximum density of the true linear regression equations in each leaf fixed at 5. This gives trees with complicated regression functions and varying complexities of hyperplane splits. The results are shown in Figure 5-7. We can see that when the split density is 1 and the tree has parallel splits only, the results are similar to the prior results for regression density 5 in Figure 5-6. We also see that ORT-LH performs the similarly to ORT-L, indicating that ORT-LH is not overfitting despite allowing for hyperplane splits. As the split density increases, the problem becomes more difficult and the performance of all methods drops. However, the two hyperplane methods, ORT-H and ORT-LH, decrease more slowly in performance that the others. As expected, the strongest performing method is ORT-LH, indicating that it is able to effectively learn the structure of the trees with linear predictions and hyperplane splits.

For the second test, we fix the maximum split density to 3 and vary the density of the regression predictions in each leaf. Figure 5-8 shows the results. Again we see that ORT-LH is the strongest performing method, followed by ORT-H. The results

Figure 5-8: Synthetic experiments showing the effect of regression density.



are largely the same across all regression densities, which combined with the previous test would indicate that the primary determinant of performance in this setting is the complexity of the splits in the tree rather than the complexity of the predictions in each leaf.

Summary

We conclude this section with a summary of the results across all the synthetic experiments conducted.

First and most importantly, we saw that the different classes of trees were able to correctly learn the structure of the ground truth trees of the same type, meaning that ORT-L could correctly learn when the truth had linear predictions, and ORT-LH performed well when the truth had both linear predictions and hyperplanes. This is clear evidence that our training procedure is able to effectively train trees of the class we are seeking.

Secondly, we found that the increased flexibility of trees with linear predictions and/or hyperplane splits did not lead them to overfit in scenarios where the true model was of a simpler class. For instance, ORT-L did not overfit when the true predictions were constant, and ORT-LH did not overfit when the ground truth tree had just parallel splits and linear predictions. This demonstrates that there is no concern with choosing a model class that is "too powerful" for a given dataset; the only tradeoff is the increased computational time.

Finally, we found that our methods compared favorably to both random forests and boosted trees in all scenarios, particular as the ground truth models became more complicated.

5.4 Experiments with Real-World Datasets

In this section, we evaluate the performance of Optimal Regression Trees with Linear Predictions, both with and without hyperplane splits, on the same set of real-world datasets as in Section 4.5. This will allow us to evaluate the impact of allowing for the more complicated linear predictions in each leaf of the tree in a variety of practical examples.

Experimental Setup

The experimental setup and sample of 26 datasets was the same for these experiments as in Section 2.6. We added ORT-L and ORT-LH to the pool of methods with $R = 100$ random restarts, $H = 5$ random hyperplane restarts and `minbucket` $= 1$, with the maximum depth D and regularization parameter λ tuned as described in Section 5.3.

Results

In Figure 5-9 we present the mean out-of-sample R^2 across the 26 for each method, as a function of the maximum tree depth allowed. Note that compared to Figure 4-17 the only difference is the addition of the ORT-L and ORT-LH methods.

We can see that the two new methods perform very strongly. ORT-L has higher performance than boosting at low depths, but is overtaken by boosting as the depth increases. At depth 10, the performance of ORT-L is roughly equivalent to random forests and performs slightly worse than boosted trees. ORT-LH is the strongest performing method at small depths, and at larger depths performs comparably to boosted trees, ahead of both random forests and ORT-L.

Table 5.1: Performance results for regression methods (with maximum depth 10) on each of the 26 datasets. For each method, we show the out-of-sample R^2 followed by the method's rank on the dataset in parentheses. A rank of 1 indicates the method had the best performance on a dataset and a rank of 5 indicates the method performed worst.

Dataset	CART	ORT	ORT-H	ORT-L	ORT-LH	RF	Boosting
abalone	0.460 (6)	0.457 (7)	0.519 (4)	0.545 (2)	0.549 (1)	0.477 (5)	0.536 (3)
ailerons	0.746 (6)	0.736 (7)	0.806 (5)	0.845 (1)	0.836 (3)	0.832 (4)	0.844 (2)
airfoil-self-noise	0.816 (6)	0.814 (7)	0.863 (5)	0.893 (4)	0.896 (3)	0.910 (2)	0.941 (1)
auto-mpg	0.771 (5)	0.771 (6)	0.766 (7)	0.845 (3)	0.838 (4)	0.847 (2)	0.849 (1)
automobile	0.581 (6)	0.590 (5)	0.495 (7)	0.688 (3)	0.631 (4)	0.732 (2)	0.736 (1)
cart-artificial	0.948 (3.5)	0.948 (3.5)	0.948 (6)	0.948 (1)	0.948 (2)	0.945 (7)	0.948 (5)
communities-and-crime	0.409 (6)	0.507 (5)	0.390 (7)	0.721 (2)	0.749 (1)	0.702 (3)	0.679 (4)
computer-hardware	0.789 (7)	0.858 (5)	0.797 (6)	0.955 (2)	0.973 (1)	0.929 (3)	0.929 (4)
concrete-compressive	0.453 (6)	0.364 (7)	0.569 (5)	0.862 (2)	0.873 (1)	0.690 (4)	0.792 (3)
concrete-flow	0.208 (7)	0.251 (6)	0.380 (5)	0.461 (2)	0.476 (1)	0.437 (3)	0.396 (4)
concrete-slump	0.172 (7)	0.217 (6)	0.570 (1)	0.260 (5)	0.277 (3)	0.359 (2)	0.272 (4)
cpu-act	0.970 (6)	0.966 (7)	0.975 (5)	0.985 (2)	0.984 (3)	0.981 (4)	0.985 (1)
cpu-small	0.959 (6)	0.959 (7)	0.967 (5)	0.974 (4)	0.974 (2)	0.974 (3)	0.979 (1)
elevators	0.686 (7)	0.689 (6)	0.874 (4)	0.899 (2)	0.912 (1)	0.815 (5)	0.877 (3)
friedman-artificial	0.846 (7)	0.852 (6)	0.920 (4)	0.954 (2)	0.956 (1)	0.896 (5)	0.952 (3)
housing	0.705 (7)	0.750 (5)	0.753 (4)	0.748 (6)	0.804 (3)	0.852 (2)	0.867 (1)
hybrid-price	0.512 (5)	0.464 (7)	0.509 (6)	0.620 (2)	0.648 (1)	0.615 (3)	0.577 (4)
kin8nm	0.442 (7)	0.498 (6)	0.752 (3)	0.736 (4)	0.851 (1)	0.659 (5)	0.776 (2)
lpga-2008	0.594 (7)	0.624 (6)	0.647 (5)	0.863 (1)	0.851 (2)	0.770 (4)	0.775 (3)
lpga-2009	0.802 (6)	0.786 (7)	0.848 (5)	0.880 (4)	0.904 (1)	0.890 (2)	0.885 (3)
parkinsons-motor	0.159 (6)	0.159 (7)	0.241 (4)	0.235 (5)	0.281 (3)	0.333 (2)	0.351 (1)
parkinsons-total	0.144 (7)	0.163 (6)	0.283 (4)	0.249 (5)	0.316 (3)	0.334 (2)	0.361 (1)
vote-for-clinton	0.315 (6)	0.286 (7)	0.354 (5)	0.377 (4)	0.395 (3)	0.408 (1)	0.404 (2)
wine-quality-red	0.286 (6)	0.263 (7)	0.299 (5)	0.340 (3)	0.304 (4)	0.429 (1)	0.394 (2)
wine-quality-white	0.279 (7)	0.282 (6)	0.300 (5)	0.353 (3)	0.349 (4)	0.442 (2)	0.497 (1)
yacht-hydrodynamics	0.988 (7)	0.991 (4)	0.991 (6)	0.992 (3)	0.991 (5)	0.994 (2)	0.996 (1)
Average rank	6.250	6.096	4.923	2.962	2.346	3.077	2.346

Figure 5-9: Mean out-of-sample R^2 for each method across all 26 datasets.

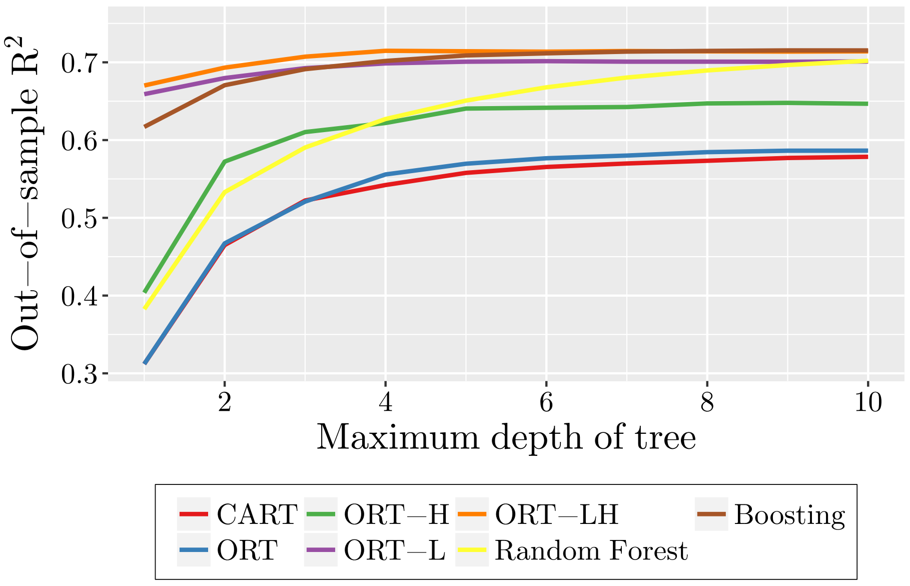


Table 5.1 presents the average out-of-sample R^2 and performance rank for each method on each dataset at the maximum depth of 10, as well as the average rank of each method. We can see that the results for CART, ORT and ORT-H mirror those in Section 4.5—ORT improves slightly upon CART and ORT-H further upon ORT, but all three are still significantly less powerful than random forests. We see that ORT-L has an average rank similar to random forests and ORT-LH performs similarly to boosting.

To quantify the significance of the differences in average rank, Table 5.2 shows the p-values for pairwise comparisons between each pair of methods, appropriately adjusted to account for multiple comparisons as described in Section 3.5. We can see that there are two groups of methods that have statistically indistinguishable performance: CART/ORT and ORT-L/ORT-LH/Random Forest/Boosting. In particular, the two strongest methods ORT-LH and boosting have an unadjusted p-value of around 0.98, indicating they are nearly identical in performance.

Together, these results on real-world datasets give evidence that allowing for linear regression models in the leaves of the regression trees provides a significant and practical increase in performance. The methods with linear models in the leaves, ORT-L and ORT-LH, significantly outperform their counterparts with only constant

Table 5.2: Pairwise significance tests for performance of regression methods (with maximum depth 10) across 26 datasets. The comparisons are presented in order of significance. For each comparison, the strongest-performing method is both bolded and stated first in the comparison. The p-values are found using Wilcoxon signed-rank test, and the adjusted p-values are calculated using the Holm-Bonferroni method to account for multiple comparisons. All results above the dividing line are significant at the 95% level.

Comparison	p-value	Adjusted p-value
ORT-L vs. CART	$\sim 10^{-8}$	$\sim 10^{-6}$
ORT-LH vs. CART	$\sim 10^{-8}$	$\sim 10^{-6}$
ORT-LH vs. ORT	$\sim 10^{-7}$	$\sim 10^{-6}$
Random Forest vs. CART	$\sim 10^{-7}$	$\sim 10^{-6}$
Random Forest vs. ORT	$\sim 10^{-7}$	$\sim 10^{-6}$
Boosting vs. CART	$\sim 10^{-7}$	$\sim 10^{-6}$
Boosting vs. ORT	$\sim 10^{-7}$	$\sim 10^{-6}$
ORT-L vs. ORT	$\sim 10^{-7}$	$\sim 10^{-6}$
ORT-LH vs. ORT-H	$\sim 10^{-5}$	0.0003
Boosting vs. ORT-H	$\sim 10^{-4}$	0.0004
ORT-H vs. CART	0.0002	0.0021
ORT-L vs. ORT-H	0.0010	0.0104
ORT-H vs. ORT	0.0020	0.0177
Random Forest vs. ORT-H	0.0047	0.0375
ORT-LH vs. ORT-L	0.0312	0.2185
Boosting vs. Random Forest	0.0842	0.5052
Boosting vs. ORT-L	0.2914	1.0000
ORT vs. CART	0.3666	1.0000
ORT-LH vs. Random Forest	0.5995	1.0000
Random Forest vs. ORT-L	0.9007	1.0000
Boosting vs. ORT-LH	0.9801	1.0000

predictions. Moreover, the performance of these two methods is on par with random forests and boosting, with no statistical evidence of any difference in performance between the four methods.

5.5 Conclusions

In this chapter, we extended the classical regression tree model to use linear regression models in each leaf of the tree rather than simply using a constant prediction.

We used the flexibility of MIO to design a formulation for Optimal Regression Trees with linear predictions that generates a tree with the combination of splits and regularized linear regression models that gives the best training error. This allows us to avoid having to restrict the power of the leaf regression models, and also ensures the regression models are only as complicated as necessary to explain the data well. We adapted the local search procedure from earlier chapters to efficiently train these trees, using a fast coordinate descent scheme to update the regression models in the leaves as the local search proceeds.

We conducted experiments with both synthetic and real-world datasets to measure the impact of adding linear predictions to the leaves. We found that these trees are able to correctly learn structure in data generated by piecewise-linear models, and moreover that they performed better than random forests and boosting on such datasets. The experiments with real-world datasets provided comprehensive evidence of the lift in performance resulting from the addition of linear predictions to the leaves, and across the sample of datasets, the performance of these regression trees was indistinguishable from random forests and boosting.

Overall, our computational experiments demonstrate that the addition of linear predictions to the leaves of Optimal Regression Trees leads to a significant increase in performance, resulting in methods that are competitive with the state of the art. Together, Chapters 2–5 provide evidence that Optimal Trees are practically relevant for both classification and regression, with our best approaches for each problem type performing comparably to random forests and boosted trees, without sacrificing the interpretability of a single decision tree.

Chapter 6

Optimal Prescription Trees

The proliferation in volume, quality, and accessibility of highly granular data has enabled decision makers in various domains to seek customized decisions at the individual level. This personalized decision making framework encompasses a multitude of applications. In online advertising internet companies display advertisements to users based on the user search history, demographic information, geographic location, and other available data they routinely collect from visitors of their website. Specifically targeting these advertisements by displaying them to appropriate users can maximize their probability of being clicked, and can improve revenue. In personalized medicine, we want to assign different drugs/treatment regimens/dosage levels to different patients depending on their demographics, past diagnosis history and genetic information in order to maximize medical outcomes for patients. By taking into account the heterogeneous responses to different treatments among different patients, personalized medicine aspires to provide individualized, highly effective treatments.

In this chapter, we consider the problem of prescribing the best option from among a set of predefined treatments to a given sample (patient or customer depending on context) as a function of the sample's features. We have access to observational data of the form $\{(\mathbf{x}_i, y_i, z_i)\}_{i=1}^n$, which comprises of n observations. Each data point $(\mathbf{x}_i, y_i, z_i)$ corresponds to the features $\mathbf{x}_i \in \mathbb{R}^d$ of the i^{th} sample, the assigned treatment $z_i \in [m]$, and the corresponding outcome $y_i \in \mathbb{R}$. We use $y(1), \dots, y(m)$ to denote the m "potential outcomes" resulting from applying each of the m respective treatments.

There are three key challenges for designing *personalized prescriptions* for each sample as a function of their observed features:

1. While we have observed the outcome of the administered treatment for each sample, we have not observed the *counterfactual outcomes*, that is the outcomes that would have occurred had another treatment been administered. Note that if this information was known, then the prescription problem reduces to a standard multi-class classification problem. We thus need to infer the counterfactual outcomes.

2. The vast majority of the available data is observational in nature as opposed to data from randomized trials. In a randomized trial, different samples are randomly assigned different treatments, while in an observational study, the assignment of treatments potentially, and often, depends on features of the sample. Different samples are thus more or less likely to receive certain treatments and may have different outcomes than others that were offered different treatments. Consequently, our approach needs to take into account the bias inherent in observational data.

3. Especially for personalized medicine, the proposed approach needs to be interpretable, that is easily understandable by humans. Even in high speed online advertising, one needs to demonstrate that the approach is fair, appropriate, and does not discriminate people over certain features such as race, gender, age, etc. In our view interpretability is highly desirable always, and a necessity in many contexts.

We seek a function $\tau : \mathbb{R}^d \rightarrow [m]$ that selects the best treatment $\tau(\mathbf{x})$ out of the m options given the sample features $\mathbf{x}$. In doing so, we need to be both "optimal" and "accurate". We thus consider two objectives:

1. Assuming that smaller outcomes y are preferable (for example, sugar levels for personalized diabetes management), we want to minimize $E[y(\tau(\mathbf{x}))]$, where the expectation is taken over the distribution of outcomes for a given treatment

policy $\tau(\mathbf{x})$. Given that we only have data, we rewrite this expectation as

$$\sum_{i=1}^{n}\left(y_i\mathbb{I}[\tau(\mathbf{x}_i)=z_i]+\sum_{t\neq z_i}\hat{y}_i(t)\mathbb{I}[\tau(\mathbf{x}_i)=t]\right), \tag{6.1}$$

where $\hat{y}_i(t)$ denotes the unknown counterfactual outcome that would be observed if sample i were to be assigned treatment t. We refer to the objective function (6.1) as the prescription error.

2. We further want to design treatment $\tau(\mathbf{x})$ that accurately estimates the counterfactual outcomes. For this reason, our second objective is to minimize

$$\left[\sum_{i=1}^{n}\left(y_i-\hat{y}_i(z_i)\right)^2\right], \tag{6.2}$$

that is we seek to minimize the squared prediction error for the observed data.

Given our desire for optimality and accuracy, we propose in this chapter to seek a policy $\tau(\mathbf{x})$ that optimizes a convex combination of the two objectives (6.1) and (6.2):

$$\mu\left[\sum_{i=1}^{n}\left(y_i\mathbb{I}[\tau(\mathbf{x}_i)=z_i]+\sum_{t\neq z_i}\hat{y}_i(t)\mathbb{I}[\tau(\mathbf{x}_i)=t]\right)\right]+(1-\mu)\left[\sum_{i=1}^{n}\left(y_i-\hat{y}_i(z_i)\right)^2\right], \tag{6.3}$$

where the *prescription factor* μ is a hyperparameter that controls the tradeoff between the prescription and the prediction error.

Related Literature

In this section, we present some related approaches to personalization in the literature and how they relate to our work. We present some methodological papers by researchers in statistics and operations research, followed by a few papers in the medical literature.

Learning the outcome function for each treatment

A common approach in the literature is to estimate each sample's outcome under a particular treatment, and recommend the treatment that predicts the best prognosis for that sample. Formally, this is equivalent to estimating the conditional expectation $\mathbb{E}[Y|Z = t, X = x]$ for each $t \in [m]$, and assign the treatment that predicts the lowest outcome to a sample. For instance, these conditional means could be estimated by regressing the outcomes against the covariates of samples who received treatment t separately. This approach has been followed historically by several authors in clinical research (for example [45]), and more recently by researchers in statistics [102] and operations research [19]. The online version of this problem, called the contextual bandit problem, has been studied by several authors [75, 53, 6]) in the multi-armed bandit literature [52]. These papers use variants of linear regression to estimate the treatment function for each arm all while ensuring sufficient exploration, and picking the best treatment based on the m predictions for a given sample.

In the context of personalized diabetes management, Bertsimas et al. [19] use carefully constructed $k-$nearest neighbors to estimate the counterfactuals, and prescribe the treatment option with the best predicted outcome if the expected improvement (over the status quo) exceeds a threshold δ. The parameters, k and δ, used as part of this approach are themselves learned from the data.

More generally in the fields of operations research and management science, Bertsimas and Kallus [11] consider the problem of prescribing optimal decisions by directly learning from data. In this work, they adapt powerful machine learning methods and encode them within an optimization framework to solve a wide range of decision problems. In the context of revenue management and pricing, Bertsimas and Kallus [12] consider the problem of prescribing the optimal price by learning from historical demand and other side information, but taking into account that the demand data is observational. Specifically, historical demand data is available only for the observed price and is missing for the remaining price levels.

Effectively, regress-and-compare approaches inherently encode a personalization framework that consists of a (shallow) decision tree of depth one. To see this, consider a problem with $m-$arms where this approach involves estimating functions f_i for computing the outcomes of samples that received arm i, for each $1 \leq i \leq m$. This prescription mechanism can be represented as splitting the feature space into m leaves, with the first leaf constituting all the subjects who are recommended arm 1 and so on. The $i-$th leaf is given by the region $\{x \in \mathbb{R}^d : f_i(x) < f_j(x) \; \forall j \neq i, 1 \leq j \leq m\}$. However, the individual functions f can be highly nonlinear which hurts interpretability. Additionally, using only the samples who were administered arm i to compute each f_i results in using only a subset of the training data for each of these computations and the f_i's not interacting with each other while learning, which can potentially lead to less effective decision rules.

Statistical learning based approaches

Another relatively recent approach involves reformulating this problem as a weighted multi-class classification problem based on imputed propensity scores, and using off the shelf methods/solvers available for such problems. Propensity scores are defined as the conditional probability of a sample receiving a particular treatment given his/her features [107]. Clearly, for a two arm randomized control trial, these values are 0.5 for each sample. For problems where these scores are known and two armed studies, Zhou et al. [123] propose a weighted SVM based approach to learn a classifier that prescribes one of the two treatment options. However, this analysis is restricted to settings where these scores are perfectly known and predefined in the trial design, e.g., randomized clinical trials (propensities are constant) or stratified designs (where the dependence of the treatment assignment on the covariates is known a priori).

In observational studies, these probabilities are typically not known, and hence are usually estimated via maximum likelihood estimation. However, there are multiple proposed methods for estimating these scores, e.g., using machine learning [119] or as primarily covariate balancing [67], and the choice of method is not clear a priori. Once these probabilities are known or estimated, the average outcome is computed using approaches based on the inverse probability of treatment weighting estimator.

This involves multiplying the observed outcome by the inverse of the propensity score (this approach is also referred to as importance/rejection sampling in the machine learning literature). While this method has desirable asymptotic properties and low bias, dividing the outcome by the estimated probabilities may lead to unstable, high variance estimates for small samples.

Tree based approaches

Continuing in the spirit of adapting machine learning approaches, Kallus [70] proposes personalization trees (and forests), which adapt regular classification trees [25] to directly optimize the prescription error. The key differences from our approach are that we modify our objective to account for the prediction error, and use the methodology introduced in Chapters 2–5 to design near optimal trees, which improves performance significantly. Athey and Imbens [3] and Wager and Athey [118] also use a recursive splitting procedure of the feature space to construct causal trees and causal forests, respectively, which estimate the causal effect of a treatment for a given sample, or construct confidence intervals for the treatment effects, but not explicit prescriptions or recommendations which is the main point of the current paper. Also, causal trees (or forests) are designed exclusively for studies comparing binary treatments. Additional methods that build on causal forests are proposed in the recent work of Powers et al. [101], who develop nonlinear methods, e.g., causal MARS (Multivariate Additive Regression Splines), designed to provide better estimates of the personalized average treatment effect in high dimensional problems. The causal MARS approach uses nonlinear functions, which are added to the basis in a greedy manner, as regressors for predicting outcomes via linear regression for each arm, but uses a common set of basis functions for both arms.

One of the advantages of these recent approaches—weighted classification or tree based methods—over regress and compare based approaches is that they use all of the training data rather than breaking down the problem into m (where m is the number of arms) subproblems, each using a separate subset of the data. This key modification increases the efficiency of learning, which results in better estimates of personalized treatment effects for smaller sizes of the training data.

Personalization in medicine

Heterogeneity in patient response and the potential benefits of personalized medicine have also been discussed in the medical literature. As an illustration of heterogeneity in responses, a certain drug that works for a majority of individuals may not be appropriate for other subsets of patients, e.g., in general older patients tend to have poor outcomes independent of any treatment [79]. In an example of breast cancer, Gort et al. [54] find that even when patients receive identical treatments, heterogeneity of the disease at the molecular level may lead to varying clinical outcomes. Thus, personalized medicine can be thought of as a framework for utilizing all this information, past data, and patient level characteristics to develop a rule that assigns treatments best suited for each patient. These treatment rules have provided high quality recommendations, e.g., in cystic fibrosis [46] and mental illness [68], and can potentially lead to significant improvements in health outcomes and reduce costs.

Contributions

We propose an approach that generalizes the methods for predictive trees in Chapters 2–5 to create prescriptive trees that are interpretable, highly scalable, generalizable to multiple treatments, and either outperform out of sample or are comparable with several state of the art methods on synthetic and real world data. Specifically, the key characteristics of our method include:

- **Interpretability**: Decision trees are highly interpretable. The trees produced by our approach are highly interpretable and provide intuition on the important features that lead to a sample being assigned a particular treatment.
- **Scalability**: Similarly to predictive trees, prescriptive trees scale to problems with n in 100,000s and d in the 10,000s in seconds when they use constant predictions in the leaves and in minutes when they use a linear model.

- **Generalizable to multiple treatments**: Prescriptive trees can be applied with multiple treatments. An important desired characteristic of a prescriptive algorithm is its generalizability to handle the case of more than two possible arms. As an example, a recent review by Baron et al. [5] found that almost 18% of published randomized control trials (RCTs) in 2009 were multi-arm clinical trials, where more than two new treatments are tested simultaneously. Multi-arm trials are attractive as they can greatly improve efficiency compared to traditional two arm RCTs by reducing costs, speeding up recruitments of participants, and most importantly, increasing the chances of finding an effective treatment [98]. On the other hand, two arm trials can force the investigator to make potentially incorrect series of decisions on treatment, dose or assessment duration [98]. Rapid advances in technology have resulted in almost all diseases having multiple drugs at the same stage of clinical development, e.g., 771 drugs for various kinds of cancer are currently in the clinical pipeline [33]. This emphasizes the importance of methods that can handle trials with more than two treatment options.

- **Highly effective prescriptions**: In a series of experiments with real and synthetic data, we demonstrate that prescriptive trees either outperform out of sample or are comparable with several state of the art methods on synthetic and real world data.

Given their combination of interpretability, scalability, generalizability and performance, it is our belief that prescriptive trees are an attractive alternative for personalized decision making.

In this chapter, we describe the methodology for creating optimal prescriptive trees (OPT) and present improvements to this OPT methodology using improved counterfactual estimates. We also provide evidence of the benefits of this method with the help of synthetic and real world examples.

6.1 Optimal Prescriptive Trees

In this section, we motivate and present the OPT algorithm that trains prescriptive trees to directly minimize the objective presented in Problem (6.3) using a decision rule that takes the form of a *prescriptive tree*; that is, a decision tree that in each leaf prescribes a common treatment for all samples that are assigned to that leaf of the tree. Our approach is to estimate the counterfactual outcomes using this prescriptive tree during the training process, and therefore jointly optimize the prescription and the prediction error.

Optimal Prescriptive Trees with Constant Predictions

Observe that a decision tree divides the training data into neighborhoods where the samples are similar. We propose using these neighborhoods as the basis for our counterfactual estimation. More concretely, we will estimate the counterfactual $\hat{y}_i(t)$ using the outcomes y_j for all samples j with $z_j = t$ that fall into the same leaf of the tree as sample i. An immediate method for estimation is to simply use the mean outcome of the relevant samples in this neighborhood, giving the following expression for $\hat{y}_i(t)$:

$$\hat{y}_i(t) = \frac{1}{|\{j : \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t\}|} \sum_{j:\mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t} y_j, \tag{6.4}$$

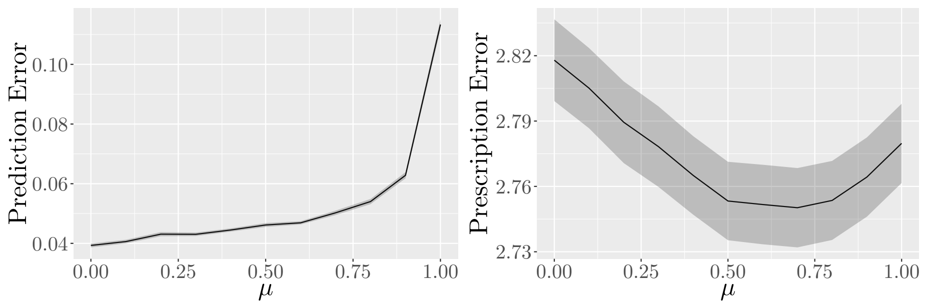
where $\mathcal{X}_{l(i)}$ is the leaf of the prescription tree into which $\mathbf{x}_i$ falls.

Substituting this back into Problem (6.3), we want to find a prescriptive tree τ that solves the following problem:

$$\begin{aligned}\min_{\tau(.)} \quad & \mu \left[\sum_{i=1}^{n} \left(y_i \mathbb{I}[\tau(\mathbf{x}_i) = z_i] + \sum_{t \neq z_i} \frac{\sum_{j:\mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t} y_j}{|\{j : \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t\}|} \mathbb{I}[\tau(\mathbf{x}_i) = t] \right)\right] \\ & + (1-\mu) \left[\sum_{i=1}^{n} \left(y_i - \frac{1}{|\{j : \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = z_i\}|} \sum_{j:\mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = z_i} y_j \right)^2\right].\end{aligned} \tag{6.5}$$

We note that when $\mu = 1$, we obtain the same objective function as Kallus [70],

Figure 6-1: Test prediction and personalization error as a function of μ



which means this objective is an unbiased and consistent estimator for the prescription error. We note that in this work they attempted to solve Problem (6.5) to global optimality using a MIO formulation based on an earlier version of Optimal Trees [10]. This approach did not scale beyond shallow trees and small datasets, and so they resorted to using a greedy CART-like heuristic to solve the problem instead. The approach we describe, using the local search approach presented in earlier chapters of the thesis, is practical and scales to large datasets while solving in tractable times. When $\mu = 0$, we obtain the objective function in Bertsimas and Dunn [10] that emphasizes prediction.

Empirically, when $\mu = 1$, we have observed that the resulting prescriptive trees lead to a high predictive error and an optimistic estimate of the prescriptive error that is not supported in out of sample experiments. Allowing μ to vary ensures that the tree predictions lead to a significant improvement of the out of sample predictive and prescriptive error.

To illustrate this observation, Figure 6-1 shows the average prediction and prescription errors as a function of μ for one of the synthetic experiments we conduct in Section 6.2. We see that using $\mu = 1$ leads to very high prediction errors, as the prescriptions are learned without making sure the predicted outcomes are close to reality. More interestingly, we see that the best prescription error is not achieved at $\mu = 1$. Instead, varying μ leads to improved prescription error, and for this particular example the lowest error is attained for μ in the range 0.5–0.8. This gives clear evidence that our choice of objective function is crucial for delivering better prescriptive trees.

Training Prescriptive Trees

We apply the Optimal Trees framework to solve Problem (6.5) and find Optimal Prescriptive Trees. The core of the algorithm remains as described in Sections 2.3 and 3.2 for parallel and hyperplane splits, respectively, and we set Problem (6.5) as the loss function to be optimized in 2.25. When evaluating the loss at each step of the coordinate descent, we calculate the estimates of the counterfactuals by finding the mean outcome for each treatment in each leaf among the samples in that leaf that received that treatment using Equation (6.4). We determine the best treatment to assign at each leaf by summing up the outcomes (observed or counterfactual as appropriate) of all samples for each treatment, and then selecting the treatment with the lowest total outcome in the leaf. Finally, we calculate the two terms of the objective using the means and best treatments in each leaf, and add these terms with the appropriate weighting to calculate the total objective value.

In addition to the existing hyperparameters (n_{min}, D_{max}, α) we introduce the following new hyperparameters:

- $n_{\text{treatment}}$: the minimum number of samples of a treatment t we need at a leaf before we are allowed to prescribe treatment t for that leaf. This is to avoid using counterfactual estimates that are derived from relatively few samples;
- μ: the prescription factor that controls the tradeoff between prescription and prediction errors in the objective function.

Optimal Prescriptive Trees with Linear Predictions

The approach detailed above trains prescriptive trees by using the mean treatment outcomes in each leaf as the counterfactual estimates for the other samples in that leaf. There is nothing special about our choice to use the mean outcome other than ease of computation, and it seems intuitive that a better predictive model for the counterfactual estimates could lead to a better final prescriptive tree. In this section, we propose using linear regression methods as the basis for counterfactual estimation inside the OPT framework.

Traditionally, tree-based models have eschewed linear regression models in the leaves due to the prohibitive cost of repeatedly fitting linear regression models during the training process, and instead have preferred to use simpler methods such as predicting the mean outcome in the leaf. However, as seen in Chapter 5, it is possible to train regression trees with linear regression models in each leaf by exploiting fast updates and coordinate descent to minimize the computational cost of fitting these models repeatedly. This provides a practical and tractable way of generating interpretable regression trees with more sophisticated prediction functions in each leaf.

We propose using this approach for fitting linear regression models from the Chapter 5 for the estimation of counterfactuals in our prescriptive trees. To do this, in each leaf we fit a linear regression model for each treatment, using only the samples in that leaf that received the corresponding treatment. We will then use these linear regression models to estimate the counterfactuals for each sample/treatment pair as necessary, before proceeding to determine the best treatment overall in the leaf using the same approach as before.

Concretely, in each leaf of the tree ℓ we fit an elastic net model for each treatment t using the relevant points in the leaf, $\{i : \mathbf{x}_i \in \mathcal{X}_\ell, z_i = t\}$, to obtain regression coefficients $\boldsymbol{\beta}_\ell^t$:

$$\min_{\boldsymbol{\beta}_\ell^t} \frac{1}{2\left|\{i : \mathbf{x}_i \in \mathcal{X}_\ell, z_i = t\}\right|} \sum_{i:\mathbf{x}_i \in \mathcal{X}_\ell, z_i = t} \left(y_i - (\boldsymbol{\beta}_\ell^t)^T \mathbf{x}_i\right)^2 + \lambda P_\alpha(\boldsymbol{\beta}_\ell^t), \tag{6.6}$$

where

$$P_\alpha(\boldsymbol{\beta}) = (1-\alpha)\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \alpha\|\vec{\beta}\|_1. \tag{6.7}$$

We proceed to estimate the counterfactuals with the following equation:

$$\hat{y}_i(t) = (\boldsymbol{\beta}_{\ell(i)}^t)^T \mathbf{x}_i, \tag{6.8}$$

where $\ell(i)$ is the leaf containing sample i. The overall objective function is therefore

$$\begin{aligned}\min_{\tau(.),\boldsymbol{\beta}} \quad & \mu\left[\sum_{i=1}^{n}\left(y_i\mathbb{I}[\tau(\mathbf{x}_i)=z_i]+\sum_{t\neq z_i}(\boldsymbol{\beta}^t_{\ell(i)})^T\mathbf{x}_i\mathbb{I}[\tau(\mathbf{x}_i)=t]\right)\right] \\ & +(1-\mu)\left[\sum_{i=1}^{n}\left(y_i-(\boldsymbol{\beta}^t_{\ell(i)})^T\mathbf{x}_i\right)^2+\lambda\sum_{t=1}^{m}\sum_{\ell}P_\alpha(\boldsymbol{\beta}^t_\ell)\right],\end{aligned} \tag{6.9}$$

where the regression models $\boldsymbol{\beta}$ are found by solving the elastic net problems (6.6) defined by the prescriptive tree. Note that we have included the elastic net penalty in the prediction accuracy term, mirroring the structure of the elastic net problem itself. This is so that our objective accounts for overfitting the $\boldsymbol{\beta}$ coefficients in the same way as standard regression. We solve this problem using the approach from 5.2, modified to fit a regression model for each treatment in each leaf, rather than just a single regression model per leaf.

There are two additional hyperparameters in this model over OPT, namely the degree of regularization in the elastic net λ and the parameter α controlling the trade-off between ℓ_1 and ℓ_2 norms in (6.7). We have found that we obtain strong results using only the ℓ_1 norm, and so this is what we use in all experiments. We select the regularization parameter λ through validation.

6.2 Experiments with Synthetic Datasets

In this section, we design simulations on synthetic datasets to evaluate and compare the performance of our proposed methods with other approaches. Since the data set is simulated, the counterfactuals are fully known, which enables us to compare with the ground truth. In the remainder of this section, we present our motivation behind these experiments, describe the data generating process and the methods we compare, followed by computational results and conclusions.

Motivation

The general motivation of these experiments is to investigate the performance of the OPT method for various choices of synthetic data. Specifically, as part of these experiments, we seek to answer the following questions.

1. *How well does each method prescribe, i.e., compute the decision boundary* $\{\mathbf{x} \in \mathbb{R}^d : y_0(\mathbf{x}) = y_1(\mathbf{x})\}$*?*

2. *How accurate are the predicted outcomes?*

Experimental Setup

Our experimental setup is motivated by that shown in Powers et al. [101]. We generate n data points $\mathbf{x}_i, i = 1, \ldots, n$ where each $\mathbf{x}_i \in \mathbb{R}^d$. Each $\mathbf{x}_i$ is generated i.i.d. such that the odd numbered coordinates j are sampled from $x_{ij} \sim \text{Normal}(0, 1)$, while the even numbered coordinates j are sampled from $x_{ij} \sim \text{Bernouilli}(0.5)$.

Next, we simulate the observed outcomes under each treatment. We restrict the scope of these simulations to two treatments (0 and 1) so that we can include in our comparison methods those that only support two treatments. For each experiment, we define a *baseline* function that gives the base outcome for each observation and an *effect* function that models the effect of the treatment being applied. Both of these are functions of the covariates $\mathbf{x}$, centered and scaled to have zero mean and unit variance. The outcome y_t under each treatment t as a function of $\mathbf{x}$ is given by

$$y_0(x) = \texttt{baseline}(\mathbf{x}) - \frac{1}{2}\texttt{effect}(\mathbf{x}),$$
$$y_1(x) = \texttt{baseline}(\mathbf{x}) + \frac{1}{2}\texttt{effect}(\mathbf{x}).$$

Finally, we assign treatments to each observation. In order to simulate an observational study, we assign treatments based on the outcomes for each treatment so that treatment 1 is typically assigned to observations with a large outcome under treatment 0, which are likely to realize a greater benefit from this prescription. Concretely, we assign treatment 1 with the following probability:

$$\mathbb{P}(Z = 1|\mathbf{X} = \mathbf{x}) = \frac{e^{y_0(\mathbf{x})}}{1 + e^{y_0(\mathbf{x})}}.$$

In the training set, we add noise $\epsilon_i \sim \text{Normal}(0, \sigma^2)$ to the outcomes y_i corresponding to the selected treatment.

We consider three different experiments with different forms for the baseline and effect functions and differing levels of noise:

1. The first experiment has low noise, $\sigma = 0.1$, a linear baseline function, and a piecewise constant effect function:

$$\begin{aligned}\texttt{baseline}(\mathbf{x}) &= x_1 + x_3 + x_5 + x_7 + x_8 + x_9 - 2,\\ \texttt{effect}(\mathbf{x}) &= 5\mathbb{1}(x_1 > 1) - 5.\end{aligned}$$

2. The second experiment has moderate noise, $\sigma = 0.2$, a constant baseline function, and a piecewise linear effect function:

$$\begin{aligned}\texttt{baseline}(\mathbf{x}) &= 0,\\ \texttt{effect}(\mathbf{x}) &= 4\mathbb{1}(x_1 > 1)\mathbb{1}(x_3 > 0) + 4\mathbb{1}(x_5 > 1)\mathbb{1}(x_7 > 0) + 2x_8x_9.\end{aligned}$$

3. The third experiment has high noise, $\sigma = 0.5$, a piecewise constant baseline function, and a quadratic effect function:

$$\begin{aligned}\texttt{baseline}(\mathbf{x}) &= 5\mathbb{1}(x_1 > 1) - 5,\\ \texttt{effect}(\mathbf{x}) &= \frac{1}{2}(x_1^2 + x_2 + x_3^2 + x_4 + x_5^2 + x_6 + x_7^2 + x_8 + x_9^2 - 11).\end{aligned}$$

For each experiment, we generate training data with $n = 1,000$ and $d = 20$ as described above. We also generate a test set with $n = 60,000$ using the same process, without adding noise. In the test set, we know the true outcome for each observation under each treatment, so we can identify the correct prescription for each observation.

For each method, we train a model using the training set, and then use the model to make prescriptions on the test set. We consider the following metrics for evaluating the quality of prescriptions:

- *Treatment Accuracy*: the proportion of the test set where the prescriptions are correct;
- *Effect Accuracy*: the R^2 of the predicted effects, $y(1)-y(0)$, made by the model for each observation in the test set, compared against the true effect for each observation.

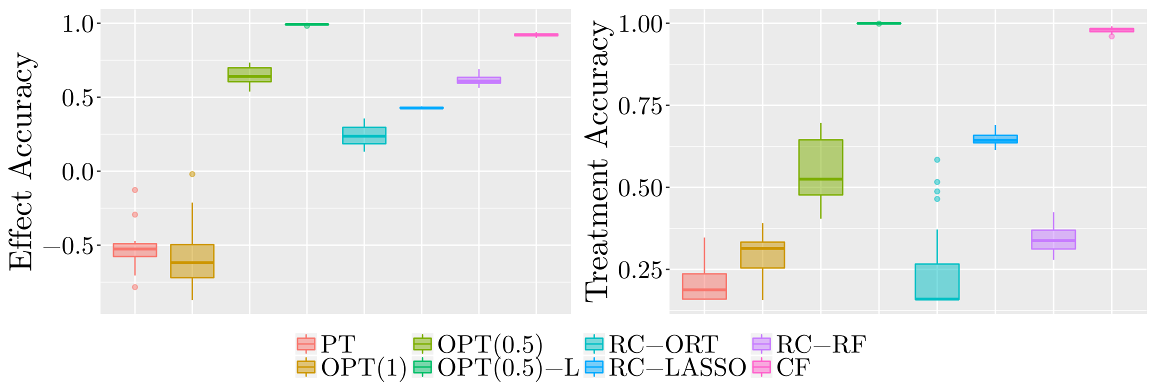
We run 100 simulations for each experiment and report the average values of treatment and effect accuracy on the test set.

Methods

We compare the following methods:

- **Prescription Trees:** We include four prescriptive tree approaches:
  - Personalization trees, denoted PT (recall that these are the same as OPT with $\mu = 1$ but trained with a greedy approach);
  - OPT with $\mu = 1$ and $\mu = 0.5$, denoted OPT(1) and OPT(0.5), respectively;
  - OPT with $\mu = 0.5$ and with linear counterfactual estimation in each leaf, denoted OPT(0.5)-L.
- **Regress-and-compare:** We include three regress-and-compare approaches where the underlying regression uses either Optimal Regression Trees (ORT), LASSO regression or random forests, denoted RC–ORT, RC–LASSO and RC–RF, respectively. For each sample in the test set, we prescribe the treatment that leads to the lowest predicted outcome.
- **Causal Methods:** We include the method of causal forests [118] with the default parameters. While causal forests are intended to estimate the individual treatment effect, we use the sign of the estimated individual treatment effect to

Figure 6-2: Performance results for Experiment 1.



determine the choice of treatment. Specifically, we prescribe 1 if the estimated treatment effect for that sample is negative, and 0, otherwise.

We also tested causal MARS on all examples, but it performed similarly to causal forests, and hence was omitted from the results for brevity.

Notice that causal forests and OPTs are joint learning methods—the training data for these approaches is the whole sample that includes both the treatment and control groups, as opposed to regress-and-compare methods which split the data and develop separate models for observations with $z = 0$ and $z = 1$.

Results

Figure 6-2 shows the results for Experiment 1. In this experiment, the boundary function is piecewise constant and the individual outcome functions are both piecewise linear. The true decision boundary is $x_1 = 1$, and the regions $\{x_1 > 1\}$ and $\{x_1 \leq 1\}$ each have constant treatment effect. The true response function in each of these regions is linear. OPT(0.5)-L outperforms all the three regress-and-compare approaches and causal forests (CF) both in treatment and effect accuracy. There is a significant improvement from OPT(0.5) to OPT(0.5)-L with the addition of linear regression in the leaves, which is unsurprising as this models exactly the truth in the data. The poorest performing method is the greedy PT, which has both low treatment accuracy, and very poor effect accuracy (note that the out of sample R^2

Figure 6-3: Tree constructed by OPT(0.5)-L for an instance of Experiment 1.

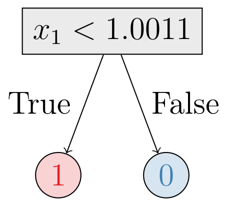
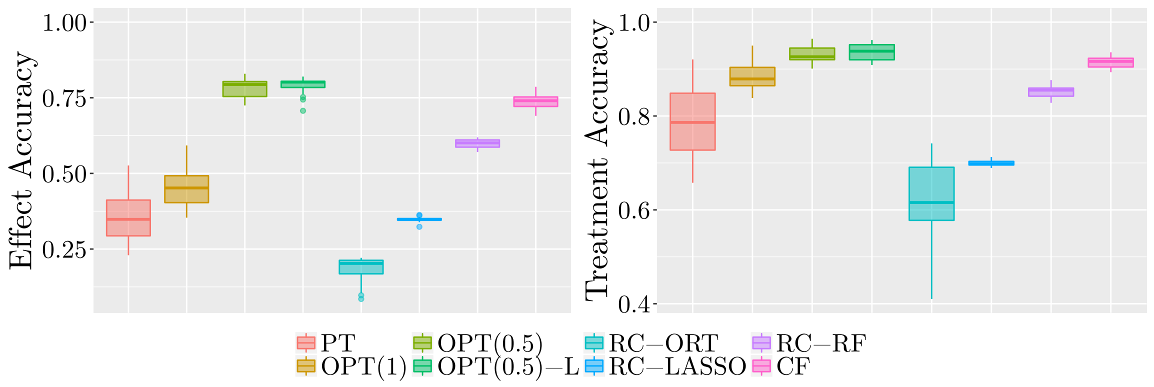


Figure 6-4: Performance results for Experiment 2.



can be negative). OPT(1) improves slightly in the treatment accuracy, but the effect accuracy is still poor. OPT(0.5) shows a large improvement in both the treatment and effect accuracies over PT and OPT(1), which demonstrates the importance of considering both the prescriptive and predictive components with the prescriptive factor μ in the objective function.

Figure 6-3 shows the tree for one of the instances of Experiment 1 under OPT(0.5)-L. Recall, the boundary function for this experiment was simply $x_1 = 1$, which is correctly identified by the tree. This particular tree has a treatment accuracy of 0.99, reflecting the accuracy of the boundary function, and an effect accuracy of 0.90, showing that the linear regressions within each leaf provide high quality estimates of the outcomes for both treatments.

The results for Experiment 2 are shown in Figure 6-4. This experiment has a piecewise linear boundary with piecewise linear individual outcome functions, with moderate noise. OPT(0.5)-L is again the strongest performing method in both treatment and effect accuracies, followed by OPT(0.5) and Causal Forests. All prescriptive tree methods have good treatment accuracy, showing that these tree models are able to effectively learn the indicator terms in the outcome functions of both arms. We again see that OPT(0.5) and OPT(0.5)-L improve upon the other tree methods, particularly in effect accuracy, as a consequence of incorporating the predictive term in the objective. The linear trends in the outcome functions of this experiment are not as strong as in Experiment 1, and so the improvement of OPT(0.5)-L over OPT(0.5) is not as large as before.

We observe that the joint learning methods perform better than the regress-and-compare methods in this example even though the outcome functions for both the treatment options do not have a common component (the baseline function is 0). We believe this is because both the methods included here, causal forests and prescriptive trees, can learn local effects effectively. Note that the structure of the boundary function is such that the function is either constant or linear in different buckets.

We plot the tree from OPT(0.5)-L for an instance of this experiment in Figure 6-5. This particular tree has a treatment accuracy of 0.925, which indicates that it has learned the decision boundary effectively, along with an effect accuracy of 0.82. We make the following observations from this tree.

1. Recall that the true boundary function for this experiment only includes the variables from x_1, x_3, x_5, x_7, x_8, and x_9, and none of the remaining variables from x_2 to x_{20}. We see that this tree does not include any of these variables as well, i.e., it has a zero false positive rate.

2. By inspecting the splits on the variables x_1, x_3, x_5 and x_7, we note that the tree has learned thresholds of close to 0 for x_3 and x_7, and 1 for x_1 and x_5, which matches with the ground truth for these variables.

3. Examining the tree more closely, we see that the prescriptions reflect the reality of which outcome is best. For example, when $x_1 \geq 0.9971$ and $x_3 \geq -0.0123$, the tree prescribes 0. This corresponds to the ground truth of the $4\mathbb{1}(x_1 > 1)\mathbb{1}(x_3 > 0)$ term becoming active, which makes it likely that treatment 1 leads to larger (worse) outcomes. We also see that the linear component in the outcome functions is reflected in the tree, as the tree assigns treatment 0 when

Figure 6-5: Tree constructed by OPT(0.5)-L for an instance of Experiment 2.

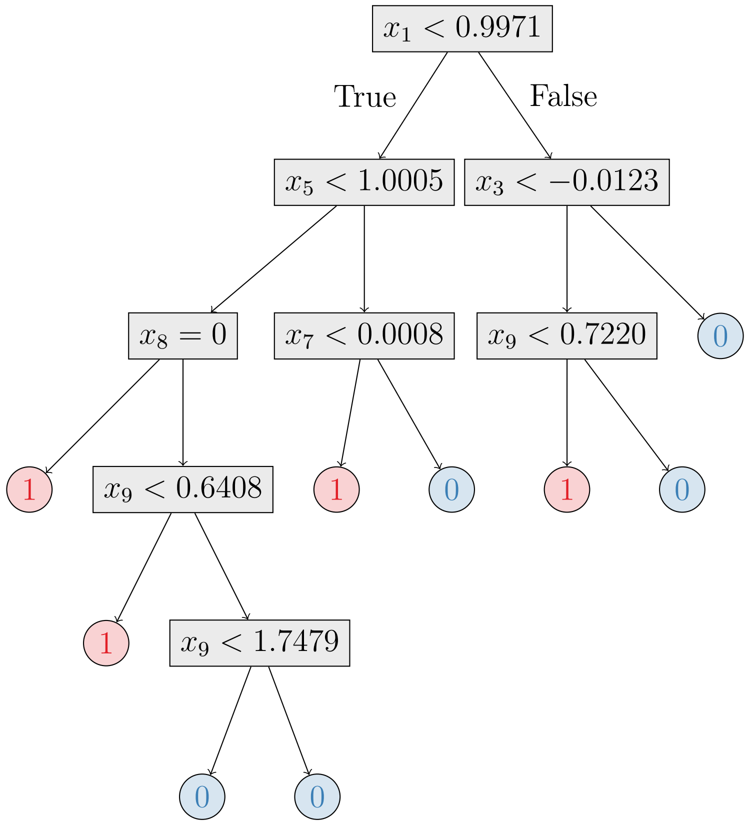
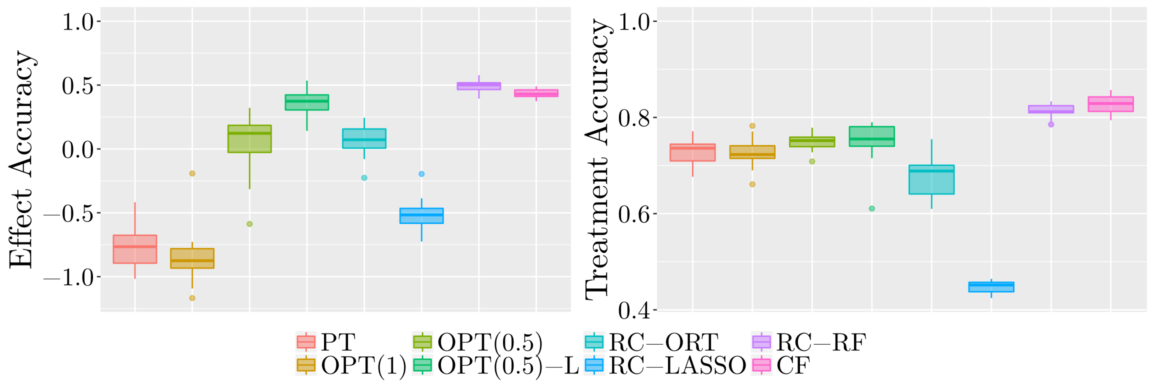


Figure 6-6: Performance results for Experiment 3.



x_9 is larger, which corresponds to the linear term in the outcome function being large.

4. Finally, we note that the tree has a split where both the terminal leaves prescribe the same treatment, which can initially seem odd. However, recall that the objective term contains both prescription and prediction errors, and a split like this can improve the prediction term in the objective, and hence the overall objective value, even though none of the prescriptions are changed.

Finally, Figure 6-6 show the results from Experiment 3. This experiment has high noise and a nonlinear quadratic boundary. Overall, regress-and-compare random forest and causal forest are the best-performing methods, followed closely by OPT(0.5)-L, demonstrating that all three methods are capable of learning complicated nonlinear relationships, both in the outcome functions and in the decision boundary. The treatment accuracy is comparable for all prescriptive tree methods, but PT and OPT(1) have very poor effect accuracy. This again demonstrates the importance of controlling for the prediction error in the objective.

In this experiment, we see that regress-and-compare random forests performs comparably to causal forests, which was not the case for the other two experiments. We believe that this is because the baseline function is relatively simple compared to the effect function, which leads to the absence of strong common trends within the two treatment outcome functions. This could make it more difficult to effectively learn from both groups jointly, mitigating the benefits of combining the groups in training.

Consequently, in this setting regress-and-compare methods could have performance closer to joint learning methods.

Summary of Synthetic Experiments

In terms of both prescriptive and predictive performance, we provide evidence that our method performs comparably with, or even outperforms the state-of-the-art methods, as evidenced by both treatment and effect accuracy metrics. Additionally, the main advantage of prescriptive trees is that they provide an explicit representation of the decision boundary, as opposed to the other methods where the boundary is only implied by the learned outcome functions. This lends credence to our claim that the trees are interpretable. In fact, from our discussion of the trees obtained for Combinations 1 and 2 in Figures 6-3 and 6-5, the trees correctly learn the true decision boundary in the data.

We also found that regress-and-compare methods that fit separate functions for each treatment are generally outperformed by joint learning methods that learn from the entire dataset. We note that if there were an infinite amount of data and the regress-and-compare methods could learn the individual outcome functions perfectly, then they would also learn the decision boundary perfectly. However, for practical problems with finite sample sizes, we have strong evidence that the performance can be significantly worse than the joint learning methods.

6.3 Experiments with Real-World Datasets

In this section, we apply prescriptive trees to some real world problems to evaluate the performance of our method in a practical setting. The first two problems, personalized warfarin dosing and personalized diabetes management, belong to the area of personalized medicine. Next, we provide personalized job training recommendations to individuals, and finally conclude with an example where we estimate the personalized treatment effect of high quality child care specialist home visits on the future cognitive test scores of infants.

Personalized Warfarin Dosing

In this section, we test our algorithm in the context of personalized warfarin dosing. Warfarin is the most widely used oral anticoagulant agent worldwide. Its appropriate dose can vary by a factor of ten among patients and hence can be difficult to establish, with incorrect doses contributing to severe adverse effects [36]. Physicians who prescribe warfarin to their patients must constantly balance the risks of bleeding and clotting. The current guideline is to start the patient at 5 mg per day, and then vary the dosage based on how the patient reacts until a stable therapeutic dose is reached [69].

The publicly available dataset we use was collected and curated by staff at the Pharmacogenetics and Pharmacogenomics Knowledge Base (PharmKGB) and members of the International Warfarin Pharmacogenetics Consortium. One advantage of this dataset is that it gives us access to counterfactuals—it contains the true stable dose for each patient found by physician controlled experimentation for $5,528$ patients. The patient covariates include demographic information (sex, race, age, weight, height), diagnostic information (reason for treatment, e.g., deep vein thrombosis etc.), pre-existing diagnoses (indicators for diabetes, congestive heart failure, smoker status etc.), current medications (Tylenol etc.), and genetic information (presence of genotype polymorphisms of CYP2C9 and VKORC1). The correct dose of warfarin was split into three dose groups: low (≤ 3 mg/day), medium (> 3 and < 5 mg/day), and high(≥ 5 mg/day), which we consider as our three possible treatments 0, 1, and 2.

Our goal is to learn a policy that prescribes the correct dose of warfarin for each patient in the test set. In this dataset, we know the correct dose for each patient, and so we consider the following two approaches for learning the personalization policy.

Personalization when counterfactuals are known

Since we know the correct treatment z_i^* for each patient, we can simply develop a prediction model that predicts the optimal treatment z given covariates $\mathbf{x}$. This is a standard multi-class classification problem, and so we can use off-the-shelf algorithms for this problem. Solving this classification problem gives us a bound on the performance of our prescriptive algorithms, as this is the best we could do if we had perfect information.

Personalization when counterfactuals are unknown

Since it is unlikely that a real world dataset will consist of these optimal prescriptions, we reassign some patients in the training set to other treatments so that their assignment is no longer optimal. To achieve this, we follow the setup of Kallus [70], and assume that the doctor prescribes warfarin dosage according to the following probabilistic assignment model:

$$\mathbb{P}(Z = t|\mathbf{X} = \mathbf{x}) = \frac{1}{S}\exp\left(\frac{(t-1)(BMI-\mu)}{\sigma}\right), \tag{6.10}$$

where μ, σ are the population mean and standard deviation of patients' BMI respectively, and the normalizing factor

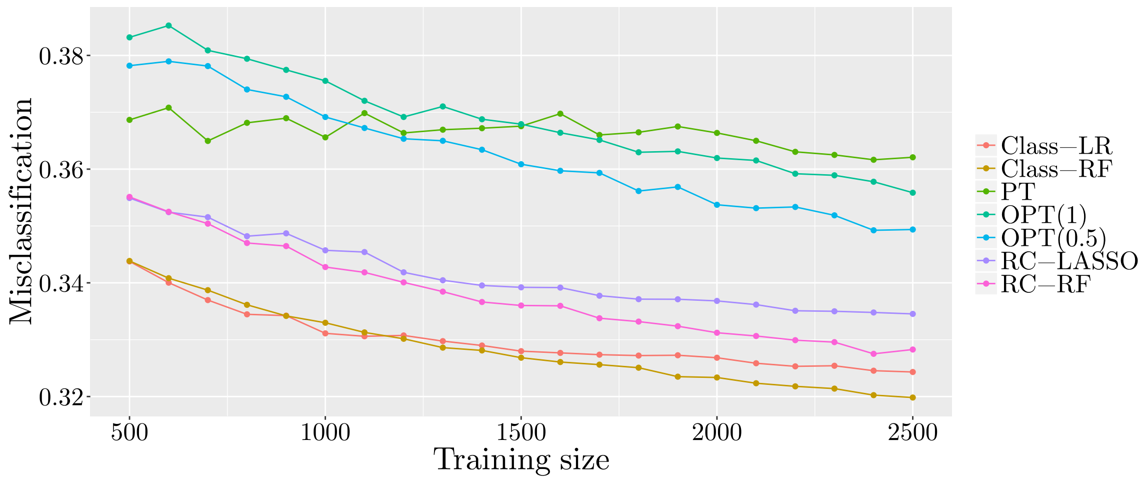
$$S = \sum_{t=1}^{3} \exp\left(\frac{(t-1)(BMI-\mu)}{\sigma}\right).$$

We use this probabilistic model to assign each patient i in the training set a new treatment $\hat{z}_i$, and then set $y_i = 0$ if $\hat{z}_i = z_i$, and $y_i = 1$, otherwise. We proceed to train our methods using the training data $(x_i, y_i, \hat{z}_i),\ i = 1, \ldots, n$. This allows us to evaluate the performance of various prescriptive methods on data which is closer to real world observational data.

Experiments

In order to test the efficacy with which our algorithm learns from observational data, we split the data into training and test sets, where we vary the size of the training set as $h = 500, 600, \ldots, 2500$, and the size of the test set is fixed as $n_{test} = 2500$. We perform 100 replications of this experiment for each n, where we re-sample the

Figure 6-7: Misclassification rate for warfarin dosing prescriptions as a function of training set size.



training and test sets of respective sizes without replacement each time. We report the misclassification (error) rate on the test set, noting that the full counterfactuals are available on the test set.

We compare the methods described in Section 6.2, but do not include OPT(0.5)-L as we did not observe any benefit when adding continuous estimates of the counterfactuals, possibly due to the discrete nature of the outcomes in the problem. We also do not include causal forests as the problem has more than two treatments. Additionally, to evaluate the performance of prescriptions when all outcomes are known, we treat the problem as a multi-class classification problem and solve using off-the-shelf algorithms. We use random forests [29], denoted Class–RF, and logistic regression, denoted Class–LR.

In Figure 6-7, we present the out-of-sample misclassification rates for each approach. We see that, as expected, the classification methods perform the best with random forests having the lowest overall error rate, reaching around 32% at $n = 2,500$. This provides a concrete lower bound for the performance of the prescriptive approaches to be benchmarked against.

The greedy PT approach has stronger performance than the OPT methods at low n, but as n increases this advantage disappears. At $n = 2,500$, OPT(1) algorithm outperforms PT by about 0.6%, which shows the improvement that is gained by solving the prescriptive tree problem holistically rather than in a greedy fashion. OPT(0.5) improves further upon this by 0.6%, demonstrating the value achieved by accounting for the prediction error in addition to the prescriptive error. The trees generated by OPT(1) and OPT(0.5) were also smaller than those from PT, making them more easily interpretable.

Finally, the regress-and-compare approaches both perform similarly, outperforming all prescriptive tree methods. We note that this is the opposite result to that found by [70], where the prescriptive trees were the strongest. We suspect the discrepancy is because they did not include random forests or LASSO as regress-and-compare approaches, only CART, k-NN, logistic regression and OLS regression which are all typically weaker methods for regression, and so the regressions inside the regress-and-compare were not as powerful, leading to diminished regress-and-compare performance. It is perhaps not surprising that the regress-and-compare approaches are more powerful in this example; they are able to choose the best treatment for *each patient* based on which treatment has the best prediction, whereas the prescription tree can only make prescriptions for *each leaf*, based on which treatment works well across all patients in the leaf. This added flexibility leads to more refined prescriptions, but at a complete loss of interpretability which is a crucial aspect of the prescription tree.

Overall, our results show that there is a significant advantage to both solving the prescriptive tree problem with a view to global optimality, and accounting for the prediction error as well as the prescription error while optimizing the tree.

Personalized Diabetes Management

In this section, we apply our algorithms to personalized diabetes management using patient level data from Boston Medical Center (BMC). This dataset consists of electronic medical records for more than 1.1 million patients from 1999 to 2014. We consider more than 100,000 patient visits for patients with type 2 diabetes during this period. Patient features include demographic information (sex, race, gender etc.), treatment history, and diabetes progression. This dataset was first considered in Bertsimas et al. [19], where the authors propose a k-nearest neighbors (kNN) regress-and-compare approach to provide personalized treatment recommendations for each patient from the 13 possible treatment regimens. We compare our prescriptive trees method to several regress-and-compare based approaches, including the previously proposed kNN approach.

We follow the same experimental design as in Bertsimas et al. [19]. The data is split 50/50 into training and testing. The models are constructed using the training data and then used to make prescriptions on the testing data. The quality of the predictions on the testing data is evaluated using a kNN approach to impute the counterfactuals on the test set—we also considered imputing the counterfactuals using LASSO and random forests and found the results were not sensitive to the imputation method. We use the same three metrics to evaluate the various methods: the mean HbA_{1c} improvement relative to the standard of care; the percentage of visits for which the algorithm's recommendations differed from the observed standard of care; and the mean HbA_{1c} benefit relative to standard of care for patients where the algorithm's recommendation differed from the observed care.

We varied the number of training samples from 1,000–50,000 (with the test set fixed) to examine the effect of the amount of training data on out-of-sample performance. We repeated this process for ten different splittings of the data into training and testing to minimize the effect of any individual split on our results.

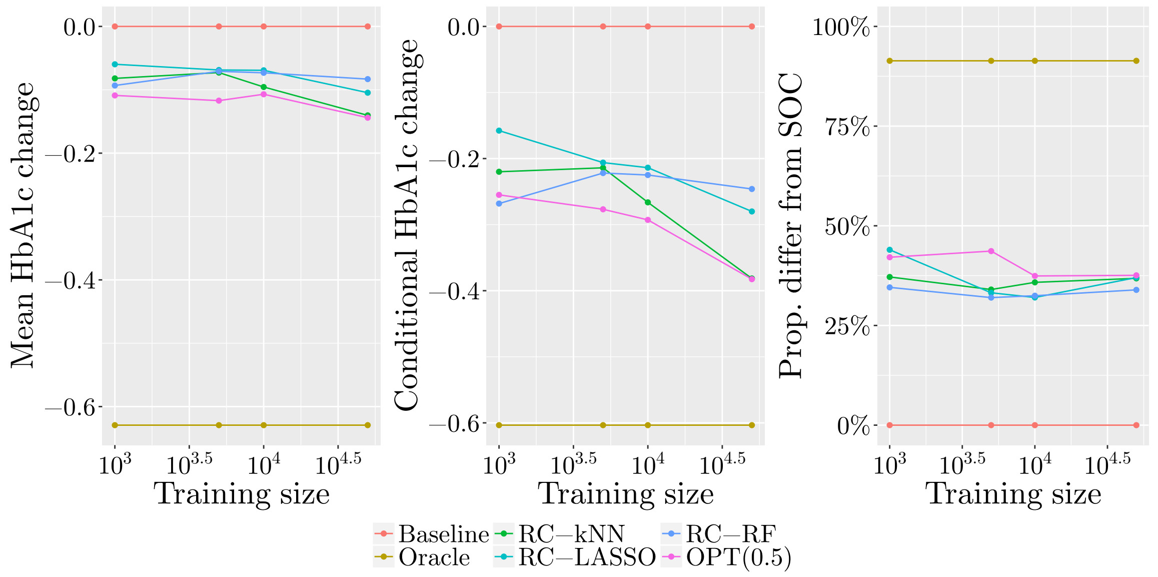
In addition to methods defined in Section 6.2, we compare the following approaches:

- **Baseline:** The baseline method continues the current line of care for each patient.

- **Oracle:** For comparison purposes, we include an oracle method that selects the best outcome for each patient using the imputed counterfactuals on the test set. This method therefore represents the best possible performance on the data.

- **Regress-and-compare:** In addition to RC–LASSO, RC–RF, we include k-nearest neighbors regress-and-compare, denoted RC–kNN, to match the approaches used in [19]

Figure 6-8: Comparison of methods for personalized diabetes management. The leftmost plot shows the overall mean change in HbA1c across all patients (lower is better). The center plot shows the mean change in HbA1c across only those patients whose prescription differed from the standard-of-care. The rightmost plot shows the proportion of patients whose prescription was changed from the standard-of-care.



The results of the experiments are shown in Figure 6-8. We see that our results for the regress-and-compare methods mirror those of [19]; RC–kNN is the best performing regression method for prescriptions, and the performance increases with more training data. RC–LASSO increases in performance with more data as well, but performs uniformly significantly worse than kNN. RC–RF performs strongly with limited data, but does not improve as more training data becomes available. OPT(0.5) offers the best performance across all training set sizes. Compared to RC–kNN, OPT(0.5) is significantly stronger at smaller training set sizes, supporting our intuition that it makes better use of the data by considering all treatments simultaneously rather than partitioning based on treatment. At higher training set sizes, the performance behaviors of RC–kNN and OPT(0.5) become similar, suggesting that the methods may be approaching the performance limits of the dataset.

These computational experiments offer strong evidence that the prescriptions of OPT are at least as strong as those from RC–kNN, and significantly stronger at smaller training set sizes. The other critical advantage is the increased interpretability of OPT compared to RC–kNN, which is itself already more interpretable than other regress-and-compare approaches. To interpret the RC–kNN prescription for a patient, one must first find the set of nearest neighbors to this point among each of the possible treatments. Then, in each group of nearest neighbors, we must identify the set of common characteristics that determine the efficacy of the corresponding treatment on this group of similar patients. When interpreting the OPT prescription, the tree structure already describes the decision mechanism for the treatment recommendation, and is easily visualizable and readily interpretable.

Personalized Job training

In this section, we apply our methodology on the Jobs dataset [74], a widely-used benchmark dataset in the causal inference literature, where the treatment is job training and the outcomes are the annual earnings after the training program. This dataset is obtained from a study based on the National Supported Work program[1]. This study consists of 297 and 425 individuals in the control and treated groups respectively, where the treatment indicator z_i is 1, if the subject received job training in 1976–77 or 0, otherwise. The dataset has seven covariates which include age, education, race, marital status, if the individual earned a degree or not, and prior earnings (earnings in 1975) and the outcome y_i is 1978 annual earnings.

We split the full dataset into 70/30 training/testing samples, and averaged the results over 1000 such splits to plot the out of sample average personalized income. Since the counterfactuals are not known for this example we again employ the nearest-neighbor matching algorithm from [19] to impute the counterfactual values on the test set. Using these imputed values, we compute the cost of policies prescribed by each method. Note that for this example, the higher the out of sample income, the better.

We include causal forests in our comparison for this problem as it only has two treatment options.

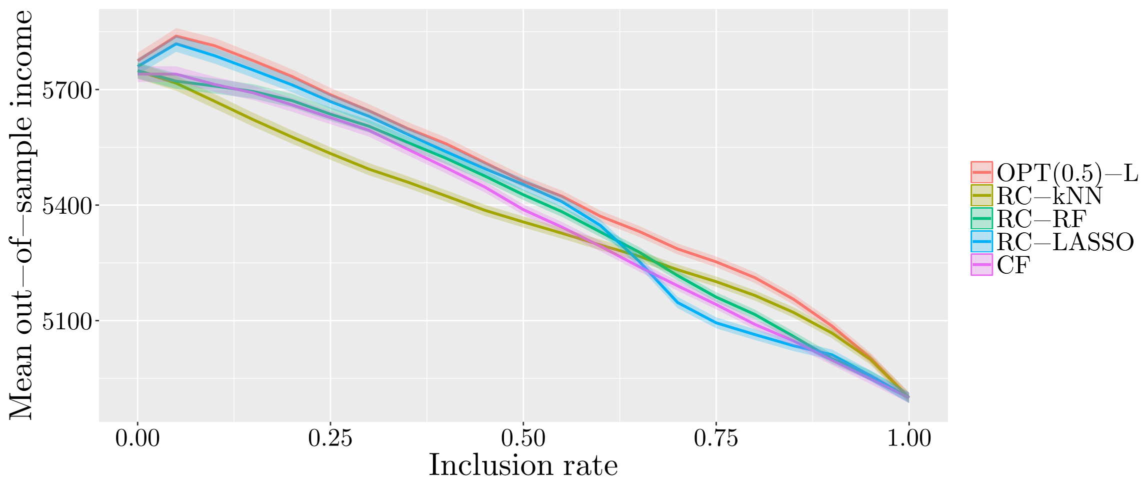
In Table 6.1, we present the average net personalized income on the test set, as prescribed by each of the five methods. For each method, we only prescribe a

[1]Available at http://users.nber.org/~rdehejia/nswdata2.html

Table 6.1: Average personalized income on the test set for various methods, arranged in increasing order.

Method	Average income ($)	Standard error ($)
Baseline	5436.58	10.81
CF	5880.70	17.92
RC–kNN	5884.28	17.79
RC–RF	5892.43	17.78
RC–LASSO	5941.39	18.94
OPT(0.5)-L	5979.84	18.07
Oracle	7697.23	17.16

Figure 6-9: Out-of-sample average personalized income as a function of inclusion rate.



treatment for an individual in the test set if the predicted treatment effect for that individual is higher than a certain value $\delta > 0$, whose value we vary and choose such that it leads to the highest possible predicted average test set income. We find the best such δ for each instance, and average the best prescription income over 1000 realizations for each method. From the results, we see that OPT(0.5)-L obtains an average personalized income of $5980, which is higher than the other methods. The next closest method is RC–LASSO, which obtains an average income of $5941.

In Figure 6-9, we present the out-of-sample incomes as a function of the fraction of subjects for which the intervention is prescribed (the inclusion rate), which we obtain by varying the threshold δ described above. We see that the average income in the test set is highest for OPT(0.5)-L at all values of the inclusion rate, indicating that our OPT method is best able to estimate the personalized treatment effect across all subjects. We also see that the income peaks at a relatively low inclusion rate, showing that we are able to easily identify a subset of the subjects with large treatment effect.

Estimating Personalized Treatment Effects for Infant Health

In this section, we apply our method for estimating the personalized treatment effect of high quality child care specialist home visits on the future cognitive test scores of children. This dataset is based on the Infant Health Development Program (IHDP) and was compiled by Hill [61]. Following Hill [61], the original randomized control trial was made imbalanced by removing a biased subset of the group that had specialist home visits. The final dataset consists of 139 and 608 subjects in the treatment and control groups respectively, with $z_i = 1$ indicating treatment (specialist home visit), and a total of 25 covariates which include child measurements such as child-birth weight, head circumference, weeks born pre-term, sex etc., along with behaviors engaged during the pregnancy—cigarette smoking, alcohol and drug consumption etc., and measurements on the mother at the time she gave birth—age, marital status, educational attainment etc.

In this example we focus on estimating the individual treatment effect, since it has been acknowledged that the program has been successful in raising test scores of treated children compared to the control group (see references in Hill [61]). The outcomes are simulated in such a way that the average treatment effect on the control subjects is positive (setting B in Hill [61] with no overlap). However, note that even though the sign and magnitude of the average treatment effect is known, there is still heterogeneity in the magnitudes of the individual treatment effects. In all our experiments, we split the data into training/test as 90/10, and compute the error of the treatment effect estimates on the test set compared to the true noiseless outcomes (known). We average this value over 100 splits of the dataset, and compare the test set performance for each method.

In Table 6.2, we present the means and standard errors of the R^2 of the personalized treatment effect estimates on the test set, given by each of the four methods. We

Table 6.2: Average R^2 on the test set for each method when estimating the personalized treatment effect.

Method	Mean R^2	Standard error
CF	0.543	0.015
RC–LASSO	0.639	0.018
RC–RF	0.704	0.013
OPT(0.5)-L	0.759	0.013

see that OPT(0.5)-L obtains the highest average R^2 value of 0.759, followed by RC-Random forests with 0.704. This again gives strong evidence that our OPT methods can deliver high-quality prescriptions whilst simultaneously maintaining interpretability.

6.4 Conclusions

In this chapter, we presented an interpretable approach for personalizing treatments that learns from observational data. Our method relies on iterative splitting of the feature space, and can handle the case of more than two treatment options. We applied this method to synthetic and real world datasets, and illustrate its superior prescriptive power compared to other state of the art methods.

Chapter 7

Diagnostics for Children After Head Trauma

Between 2006 and 2010, an estimated 750,000 emergency department visits pertained to pediatric head trauma [85, 86]. Most children with head injury have apparently minor trauma [86, 44]. In children with minor head trauma, the main challenge is to identify clinically important intracranial injuries that necessitate immediate intervention or close observation. Currently, computed tomography (CT) is the standard for the rapid diagnosis of intracranial injury [47], but it is costly, may require sedation, and exposes patients to ionizing radiation which may increase the risk of malignancies later in life [32, 31, 87]. However, patients without substantially altered mental status, e.g., with Glasgow Coma Scale (GCS) scores of 14 or 15, rarely suffer from clinically important traumatic brain injury (TBI) or have evidence of intracranial injury with CT imaging [73]. Avoiding needless CTs in such patients is highly desirable [73]. To this end, the Pediatric Emergency Care Applied Research Network (PECARN) has developed and validated rules for triaging which children with head trauma but without substantially altered mental status should receive head CT to diagnose clinically important TBI [73]. The PECARN prediction tools are widely used [122, 108] and have been independently validated multiple times [84, 66].

The PECARN tools are easy to memorize and apply, but their simplicity may come at a price in terms of maximum attainable predictive performance. Having easy-to-memorize predictive tools is desirable but hardly necessary, given the high degree of penetration of health information technologies and the wide use of clinical decision support tools in emergency departments in developed countries. In this chapter, we use the methods from Chapter 2 to develop and validate tools to predict clinically important TBI (ciTBI) that are easy to implement, and provide a performance edge on the PECARN tools [73]. The intention is to provide an example of how the improved performance of OCT can lead to materially significant impact in practice.

7.1 Data and Methods

Patients

We analyzed a prospective cohort of $42,412$ children with head trauma who were examined between June 2004 and September 2006 in emergency medicine departments participating in PECARN [73]. The mean age was 7.1 years, ranging from 0 to 18. 6263 (15%) children suffered injuries with severe mechanisms, $37,961$ (90%) had isolated head trauma, and $41,071$ (97%) had GCS score of 15.

This cohort was designed to develop and validate a classifier (predictive tool) to identify children at very low risk of ciTBI. Eligible patients presented to the emergency departments within 24 hours of a head injury. Patients who were imaged before admission, had trivial injury mechanisms, or conditions complicating assessment (e.g., known brain tumors) were excluded. Patients with GCS scores of 13 or less, ventricular shunts, or bleeding disorders were excluded. Further details about the cohort are in [73].

We followed the original analysis and stratified the cohort into $10,718$ (25%) "younger" (younger than 2 years and predominantly nonverbal) and $31,694$ (75%) "older" (older than 2 years and predominantly verbal) patient strata. The rationale was that the injury mechanisms and the patients' sensitivity to radiation and (their guardians') risk tolerance towards radiation-related side effects would likely differ in the two groups [73]. We randomly split patients into classifier training ($n = 8574$

and $n = 25,355$, respectively, in the younger and older strata) and test ($n = 2144$ and $n = 6339$, respectively, in the two strata) cohorts. Because the dataset was anonymized, we could not use the same training and test cohorts as in the original analysis [73].

Clinical outcome

The outcome of interest was ciTBI, defined *a priori* as death from TBI, neurosurgery, intubation for more than 24 hours, or hospital admission for at least 2 nights in patients with TBI-related CT findings. For detailed clinical definitions see [73].

Classifier training and test

Predictors

All predictors in the PECARN dataset were prospectively recorded. Predictors included attributes of the injury mechanism and symptoms and signs assessed at presentation (see Table 7.1). Predictors that were defined conditional on levels of other predictors were entered in the analyses as interaction effects. For example, the duration of a seizure is defined only among patients with a history of post-injury seizures, and is coded as 0 for patients with no history of seizures.

Classifiers

CART was used to derive the original PECARN rules. We developed optimal classification trees (OCT) using the methods of Chapter 2 to classify patients as having ciTBI or not. OCT was tuned to be as likely to miss a ciTBI case as the original PECARN tools are.

Test

For each classifier, we estimated the sensitivity, specificity, positive and negative predictive values (PPV and NPV, respectively), and positive and negative likelihood ratios ($LR+$ and $LR-$, respectively) in the test cohorts. For each metric, we also

Table 7.1: Prospectively collected predictors in the PECARN cohort. Listed in brackets is the coding of the predictor: px, polytomous with x levels; ox, ordinal with x levels; c, continuous. Predictors defined conditional on the presence of other predictors are indented.

History	Physical	Other
age [c]	GCS 15 [p2]	intubated [p2]
headache [p2]	altered mental status [p2]	sedated [p2]
initiation [p4]	agitated [p2]	pharmacologically paralyzed [p2]
severity [o4]	sleepy [p2]	
amnesia [p3]	slow reacting [p2]	injury mechanism (p13)
loss of conscience [p3]	repetitive questions [p2]	injury severity (o3)
duration [o4]	other [p2]	
seizures [p2]	palpable skull fracture [p3]	
initiation [p4]	depressed [p2]	
duration [o4]	basilar fracture signs [p2]	
acting normal [p2]	hemotympanum [p2]	
vomiting [p2]	CSF otorrhea [p2]	
episodes [o4]	CSF rhinorrhea [p2]	
first vomit [p4]	raccoon eyes [p2]	
last vomit[p4]	Battle's sign [p2]	
dizziness [p2]	scalp hematoma [p2]	
	size [o3]	
	location [p3]	
	supraclavicular trauma [p2]	
	face [p2]	
	neck [p2]	
	frontal scalp [p2]	
	occipital scalp [p2]	
	parietal scalp [p2]	
	temporal scalp [p2]	
	neurological deficit [p2]	
	motor [p2]	
	sensory [p2]	
	pupil reactivity [p2]	
	reflexes [p2]	
	other [p2]	
	other substantial injury [p2]	
	extremities [p2]	
	lacerations [p2]	
	cervical spine [p2]	
	torso [p2]	
	intraabdominal [p2]	
	pelvis [p2]	
	other [p2]	
	intoxication [p2]	
	bulging anterior fontanelle [p2]	

compared estimates from OCT and from the original PECARN tools, by estimating odds ratios of probability metrics and ratios of likelihood ratio metrics. Unless otherwise noted, all confidence intervals (CI) are two-sided exact 95% CIs.

Handling of missing data

All $42,412$ patients had outcome data. For each predictor in Table 7.1, we examined whether missing a value was associated with the outcome using logistic regressions. There was no evidence of strong associations between the outcome and missingness in various predictors, suggesting that data are *missing completely at random* [80]. We imputed missing values using a novel method developed in [20] that imputes the missing values to minimize the total distance of each datapoint to its K-nearest neighbors ($K = 10$ through cross-validation). Such methods have demonstrated significant improvement in imputation accuracy compared to classical (approximate) methods such as the expectation-maximization algorithm and predictive mean matching in many real-world datasets [20].

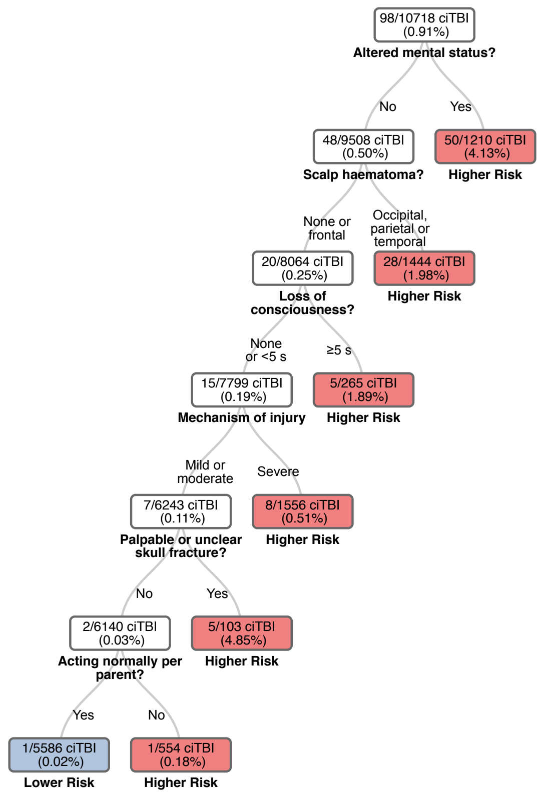
Sensitivity analyses

We also examined whether results changed when only complete cases (children with no missing values) were used. We also compared OCT against CART trees that we developed in the training dataset using the same approach as in [73]. All results from sensitivity analyses were congruent with those from the main analysis and are not shown.

7.2 Results

With the PECARN tool (Figure 7-1), patients younger than 2 years are considered at higher risk of ciTBI if they have signs of altered mental status, a palpable skull fracture, an occipital or parietal scalp hematoma, are not acting normally (per guardian), or suffered a severe mechanism of injury. Compared to the PECARN tool, the cor-

Figure 7-1: PECARN rules for ciTBI in children younger than 2 years.

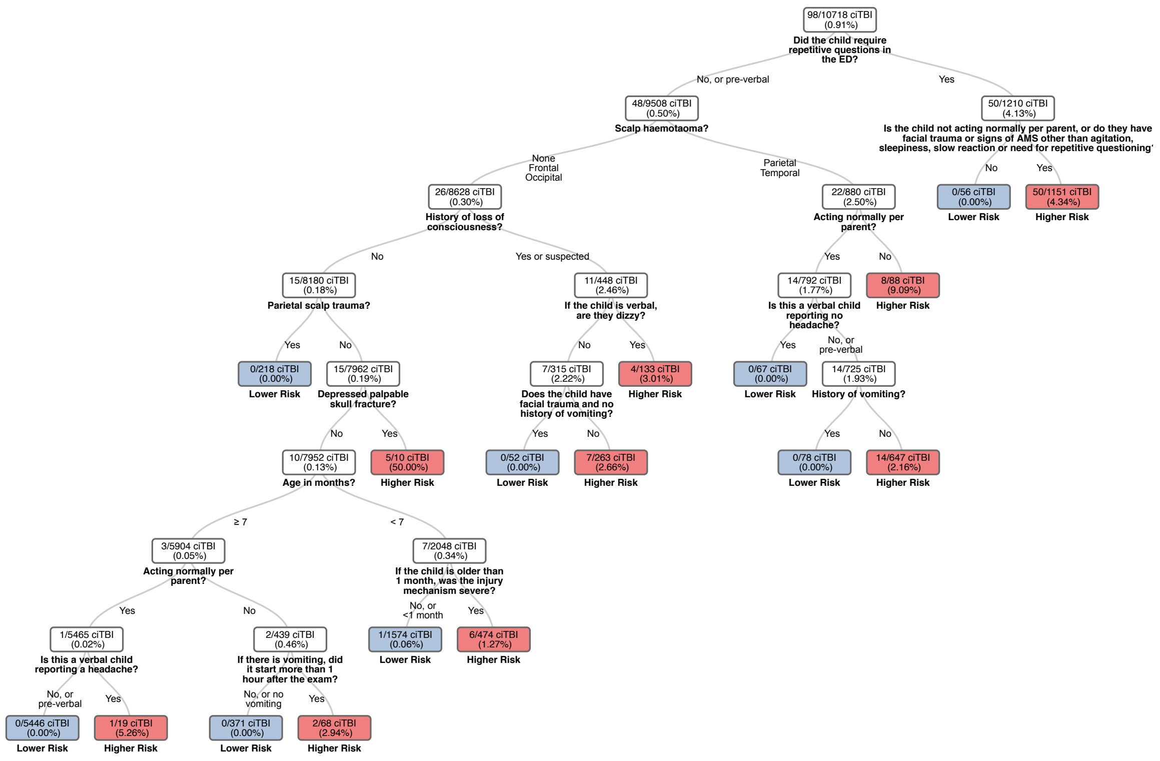


responding OCT in Figure 7-2 identified more children as being at low risk of ciTBI. For example, nonverbal children who have a parietal scalp hematoma are classified as being at higher risk of ciTBI with the PECARN tool, but they would be classified as having low risk of ciTBI by OCT if they also act normally and have no history of vomiting.

Children who are at least 2 years old are considered at higher risk of TBI with the PECARN tool if they exhibit signs of altered mental status or basilar skull fracture, have a history of loss of consciousness or of vomiting, have severe headache, or suffered a severe mechanism of injury (Figure 7-3). The corresponding OCT in Figure 7-4 identifies more children as being at lower risk, compared to the PECARN tool. For example, a child with no signs of altered mental status who only has a history of vomiting would be classified as having low risk of ciTBI if there is no supraclavicular evidence of trauma (e.g., lacerations, bruises).

The counts of children predicted to be at higher versus low risk of ciTBI with

Figure 7-2: OCT for ciTBI in children younger than 2 years.



OCT and the PECARN tools versus the true ciTBI status are shown in Tables 7.2 and 7.3 for younger and older patient strata, respectively.

In the younger stratum, both tools classified every patient with ciTBI in the same way: they identified all 17 patients with ciTBI in the validation cohort, and the same 80 out of 81 patients in the training cohort. The single patient who was mistakenly predicted to be at low risk of ciTBI by both tools was a newborn who was hospitalized for at least 2 nights and had evidence of TBI in CT imaging. That child suffered a moderate severity trauma and had no recorded suggestive signs (patient A in Table 7.4). However, OCT outperformed the PECARN rule in correctly identifying more children at low risk of ciTBI. As shown in Table 7.5, for all performance metrics, the OCT was as good or better than the PECARN tool in both cohorts.

With reference to correctly ruling out the presence of ciTBI, compared to the PECARN tool, the OCT had statistically significantly better specificity (by 21% [odds ratio 2.44] and 25% [odds ratio 3.19] in the training and test cohorts, respectively), at least as high negative predictive value, and a more favorable negative likelihood

Figure 7-3: PECARN rules for ciTBI in children at least 2 years old.

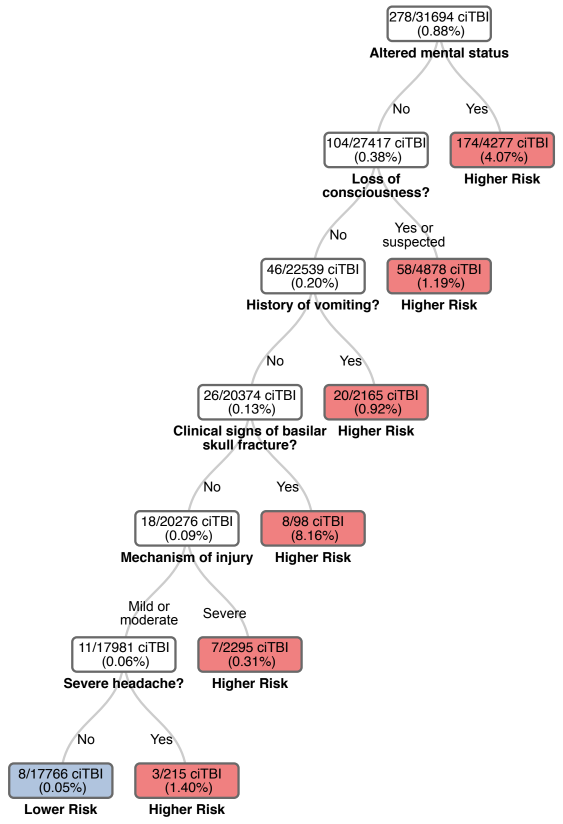
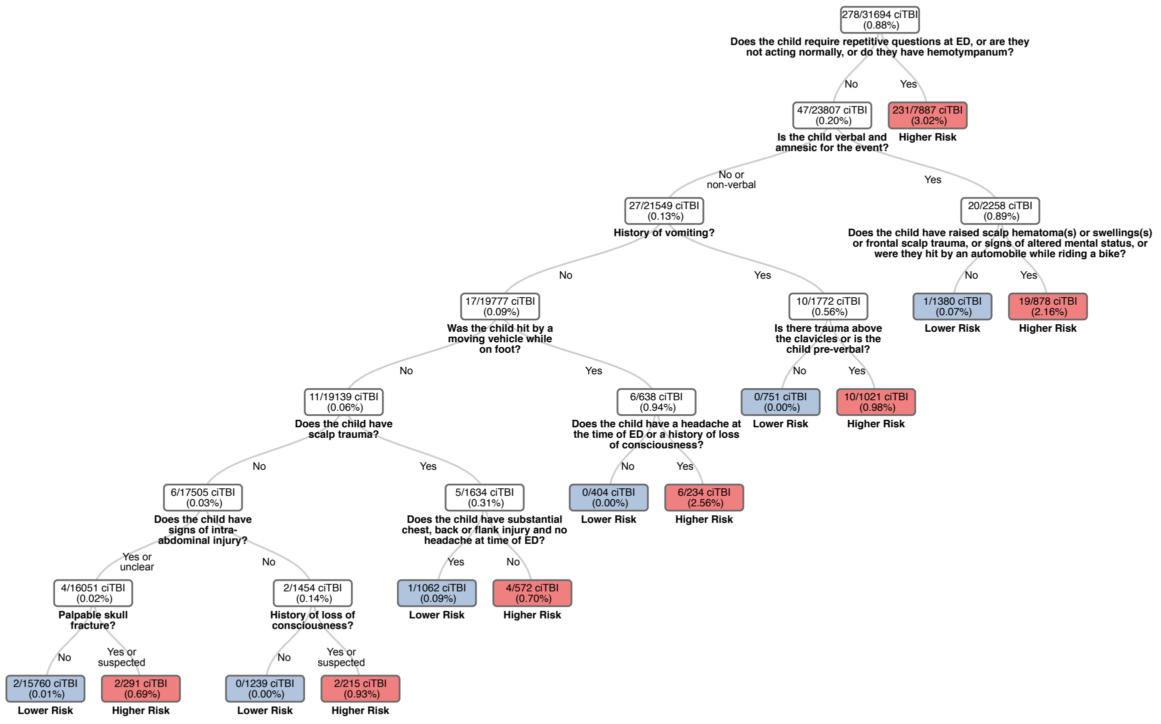


Figure 7-4: OCT for ciTBI in children at least 2 years old.



ratio (28% and 32% smaller, respectively, albeit not statistically significantly so).

In the older stratum, in the training cohort, the PECARN tool missed 7 out of 244 patients with ciTBI, and the OCT missed 3 (Table 7.4). Two patients with ciTBI were missed by both tools. One of them, a 6 year old child who fell down the stairs and presented with a small frontal hematoma, received neurosurgery. The other, a 16 year old who suffered an assault and had signs of facial trauma, was hospitalized and had TBI findings in CT imaging. The OCT (but not PECARN) misclassified a 11 year old patient who had a bike accident, and a history of brief loss of conscience, and moderate headache. The PECARN tool (but not OCT) misclassified 5 patients aged between 6 and 15 years, all with moderate severity injury mechanisms, and variable suggestive symptoms and signs (Table 7.4). Both tools classified every patient with ciTBI in the same way in the test set. They both missed a 26 month old preverbal child who was hospitalized for at least 2 nights and had evidence of TBI in CT imaging. This patient suffered a moderately severe injury falling from an elevation, and had only a medium sized parietal or temporal hematoma.

As was the case in the younger children, in children at least 2 years old, the OCT was as good or better than the PECARN tool in both cohorts for all performance metrics. Compared to the PECARN rules, OCT had statistically significantly better specificity (by 10% [odds ratio 1.54] and 3.6% [odds ratio 1.18] in the training and test cohorts, respectively), at least as high negative predictive value, and a more favorable negative likelihood ratio (54% and 5% smaller, respectively albeit not statistically significantly so; Table 7.6).

Table 7.2: Cross-classification of predictions of higher and low risk and ciTBI status in children younger than 2 years.

Cohort	ciTBI status	OCT high risk	OCT low risk	PECARN high risk	PECARN low risk
Training	Yes	80	1	80	1
	No	2300	6193	4040	4453
Test	Yes	17	0	17	0
	No	459	1668	995	1132

Table 7.3: Cross-classification of predictions of higher and low risk and ciTBI status in children 2 years and older.

Cohort	ciTBI status	OCT high risk	OCT low risk	PECARN high risk	PECARN low risk
Training	Yes	241	3	237	7
	No	9020	16091	11629	13482
Test	Yes	33	1	33	1
	No	1804	4501	2029	4276

Table 7.4: Patients with ciTBI who were missed by the PECARN or OCT rules.

ID	Cohort	Missed by	Age (verbal status)	Mechanism of injury (severity)	GCS	Recorded signs or symptoms	ciTBI
A	<2 y, dtraining	OCT, PECARN rules	Newborn (preverbal)	Object fell on head (moderate)	15	[None]	Hospitalization + TBI in CT
B	>=2 y, training	OCT, PECARN rules	6 y (verbal)	Fell down the stairs (moderate)	15	Small (<1 cm) frontal hematoma	Neurosurgery
C	>=2 y, training	OCT, PECARN rules	16 y (verbal)	Assault (moderate)	15	Facial trauma	Hospitalization + TBI in CT
D	>=2 y, training	PECARN rules	11 y (verbal)	Bike collision or fall from bike (moderate)	15	Large (>3 cm) frontal hematoma, moderate headache within 1 h, signs of injury in the flank and extremities	Hospitalization + TBI in CT
E	>=2 y, training	PECARN rules	7 y (verbal)	Wheeled transport crash other than bike or motor vehicle (moderate)	15	Medium (1–3 cm) frontal hematoma, dizziness, unclear palpation exam for skull fracture, facial trauma, neurological deficit (cranial nerve)	Hospitalization + TBI in CT
F	>=2 y, training	PECARN rules	6 y (verbal)	Wheeled transport crash other than bike or motor vehicle (moderate)	15	Small (<1 cm) parietal or temporal hematoma, moderate headache within 1 h	Hospitalization + TBI in CT
G	>=2 y, training	PECARN rules	15 y (verbal)	Sports (moderate)	15	Large (>3 cm) frontal hematoma, moderate headache within 1 h, facial trauma	Hospitalization + TBI in CT
H	>=2 y, training	PECARN rules	11 y (verbal)	Sports (moderate)	15	Amnesia for the injury, facial trauma	Hospitalization + TBI in CT
I	>=2 y, training	OCT rules	11 y (verbal)	Bike collision or fall from bike (moderate)	15	Amnesia for the injury, loss of consciousness for 1–5 mins, moderate headache	Hospitalization + TBI in CT
J	>=2 y, test	OCT, PECARN rules	26 mo (preverbal)	Fell from an elevation (moderate)	15	Medium (1–3 cm) parietal or temporal hematoma	Hospitalization + TBI in CT

Table 7.5: Predictive performance of OCT versus PECARN rules among children younger than 2 years.

Metric	OCT	PECARN	Odds Ratio
Training			
Sensitivity	98.8 (93.3, 100.0)	98.8 (93.3, 100.0)	1.00 (0.31, 3.21)
Specificity	72.9 (72.0, 73.9)	52.4 (51.4, 53.5)	2.44 (2.35, 2.54)
PPV	3.4 (2.7, 4.2)	1.9 (1.5, 2.4)	1.76 (1.68, 1.84)
NPV	100.0 (99.9, 100.0)	100.0 (99.9, 100.0)	1.39 (0.45, 4.33)
LR+	3.65 (3.50, 3.81)	2.08 (2.01, 2.15)	1.76 (1.68, 1.84)
LR-	0.02 (<0.005, 0.12)	0.02 (<0.005, 0.17)	0.72 (0.23, 2.24)
Test			
Sensitivity	100.0 (80.5, 100.0)	100.0 (80.5, 100.0)	1.00 (0.15, 6.65)
Specificity	78.4 (76.6, 80.2)	53.2 (51.1, 55.4)	3.19 (2.91, 3.50)
PPV	3.6 (2.1, 5.7)	1.7 (1.0, 2.7)	2.16 (1.82, 2.57)
NPV	100.0 (99.8, 100.0)	100.0 (99.7, 100.0)	1.47 (0.25, 8.61)
LR+	4.50 (4.02, 5.04)	2.08 (1.90, 2.27)	2.16 (1.82, 2.57)
LR-	0.04 (<0.005, 0.54)	0.05 (<0.005, 0.80)	0.68 (0.12, 4.00)

Table 7.6: Predictive performance of OCT versus PECARN rules among children at least 2 years old.

Metric	OCT	PECARN	Odds Ratio
Training			
Sensitivity	98.8 (96.4, 99.7)	97.1 (94.2, 98.8)	1.86 (0.84, 5.14)
Specificity	64.1 (63.5, 64.7)	53.7 (53.1, 54.3)	1.54 (1.50, 1.58)
PPV	2.6 (2.3, 2.9)	2.0 (1.8, 2.3)	1.31 (1.28, 1.35)
NPV	100.0 (99.9, 100.0)	99.9 (99.9, 100.0)	2.19 (1.00, 5.93)
LR+	2.75 (2.69, 2.81)	2.10 (2.04, 2.15)	1.31 (1.28, 1.35)
LR-	0.02 (0.01, 0.06)	0.05 (0.03, 0.11)	0.46 (0.17, 1.00)
Test			
Sensitivity	97.1 (84.7, 99.9)	97.1 (84.7, 99.9)	1.00 (0.30, 3.38)
Specificity	71.4 (70.3, 72.5)	67.8 (66.7, 69.0)	1.18 (1.13, 1.24)
PPV	1.8 (1.2, 2.5)	1.6 (1.1, 2.2)	1.12 (1.03, 1.23)
NPV	100.0 (99.9, 100.0)	100.0 (99.9, 100.0)	1.05 (0.34, 3.31)
LR+	3.39 (3.16, 3.64)	3.02 (2.82, 3.23)	1.12 (1.03, 1.23)
LR-	0.04 (0.01, 0.28)	0.04 (0.01, 0.30)	0.95 (0.30, 2.94)

7.3 Discussion

TBI has been deemed a 'serious public health concern' by the Centers of Disease and Prevention (CDC) [44]. In the United States, public awareness about the importance and long term implications of head injury has increased over the last years, and parents and guardians are more likely to bring children with head trauma to the emergency department for evaluation [86]. In children who have very low risk of ciTBI, avoiding superfluous CT imaging and the associated exposure to ionizing radiation reduces costs and the risk of long term radiation-induced malignancies [31, 32, 87].

We developed algorithms that outperform the PECARN tool in correctly identifying patients without ciTBI, while identifying the same patients with ciTBI among children younger than 2 years, and more patients with ciTBI among children older than 2 years. The improvements are more pronounced among children younger than 2 years, who are predominantly preverbal and thus more difficult to evaluate, may be more likely to suffer clinically important injury even when the injury mechanism is not severe, and for whom concerns about radiation exposure are greater compared to older children. Using the OCT versus PECARN would correctly reclassify 21.2% (2276/10718) of children younger than 2 years and 8.9% (2830/31694) of older children. These improvements are likely clinically important and economically consequential.

The Children's Head injury Algorithm for the prediction of Important Clinical Events (CHALICE) [41] and the Canadian Assessment of Tomography for Childhood Head Injury (CATCH) rule [95] are two other major clinical prediction rules. In a prospective external validation study in 1009 children (75% of which were older than 2.6 years), the sensitivities of PECARN, CHALICE and CATCH were 100%, 84%, and 91%, respectively, and the specificities were 61%, 85%, and 44% [42]. If the odds ratios from our head-to-head comparison between OCT and PECARN (Tables 7.5 and 7.6) transfer to the validation study of Easter et al. [42], OCT would have sensitivity around 100% and specificity between 65 and 74%.

A systematic review and decision analysis from the perspective of the National

Health Service (NHS) in England and Wales suggests that using clinical prediction rules outperforms other strategies, such as relying on isolated clinical symptoms or signs, subjecting all or none to CT imaging [97, 100]. Although the PECARN tool was the most sensitive of the examined clinical prediction rules, which included CHALICE [41], the Atabaki et al. tool [2], a University of California Davis tool [96], and the National Emergency X-Radiography Utilization Study (NEXUS) II [88], the cost-effectiveness analysis favored CHALICE and NEXUS-II over PECARN, because of their highest specificity. Our OCT models were tuned to have at least as good sensitivity as PECARN, and they attain substantially higher specificity than PECARN. Thus the herein developed OCT rules would likely be more competitive to CHALICE and NEXUS-II in the NHS's cost-effectiveness evaluation [97, 100].

However, especially in the U.S., where there is no explicit health care budget, considerations of cost-effectiveness are less directly applicable. Instead, because the uncertainty about the role of CT imaging in children with apparently minor head trauma is pervasive, emphasis is given to shared decision making between providers, patients and their families. The goal of shared decision making is to clarify what is known about the clinical utility of testing and the long term risks of the exposure to ionizing radiation during CT imaging, and to help patients and their families understand what their preferences. Empirical evidence suggests that most parents prefer disclosure of risk before proceeding with CT imaging [24]. In addition, there is evidence that the decision to proceed with CT imaging is sensitive to preferences. For example, Karpas et al. surveyed 134 parents of children older than 2 years who presented with head trauma. After a standardized education about the risks of increased exposure to ionized radiation, most parents ($n = 77$) preferred observation to immediate CT scanning, 53 preferred immediate CT, and 3 indicated no preference [71].

The OCT rules are not as easy-to-memorize as the PECARN clinical prediction rules. However, we have demonstrated that they substantially outperform the PECARN tools in the training and test cohorts. The increased classification performance may outweigh the loss in the simplicity of the tool. Having simple prediction tools is desirable but hardly necessary, considering the ease with which algorithms can be integrated in electronic medical records and other information technology systems. We found no evidence for optimism bias for OCT in extensive bootstrap-based (resampling) analyses, suggesting that these results would likely transfer to new settings. Internal validation efforts are not foolproof substitutes for completely independent empirical validation of the tool in real clinical practice, as has been done with the original PECARN tool [66, 84]. To effectively support shared decision making, a patient decision aid should be developed to help patients and parents understand what options are available to them and to clarify their values and preferences about outcomes anticipated with each option. Decision aids facilitate shared decision making by helping patients negotiate uncertainty, and make choices that best aligned with their stated and considered values. To our knowledge, such a decision aid does not exist for children with apparently mild head trauma. We believe that developing such a decision aid, and informing it with state of the science tools such as the OCT models developed here, is an important future research need.

Chapter 8

Conclusion

The primary goal of the thesis was to develop methods for constructing optimal decision trees in a practically tractable way, in order to significantly advance the state of the art in the field of decision trees. Since the 1980s, greedy heuristic methods like CART have been the decision tree method of choice in both industry and academia, and further improvements to such methods have been incremental.

Motivated by the incredible speedups in mixed-integer optimization over the past 30 years, we posed the task of creating the optimal decision tree in its natural form as a discrete optimization problem. This allowed for the entire decision tree to be created in a single step, overcoming the limitations of the classical greedy heuristics. This MIO-based approach did not scale to the problem sizes that are relevant for practical impact, and so we developed a local search procedure for efficiently optimizing the tree, in both the theoretical and practical sense. These local search methods run in very fast times and deliver high quality solutions which empirically match the globally optimal solutions from the MIO solver in all cases where this solution is known to us.

In addition to the standard axes-aligned parallel splits used by methods like CART, we also developed approaches for training trees with hyperplane splits in an efficient manner. These trees allow for greatly increased modeling power at the possible expense of some interpretability.

We first applied optimal trees to the prediction problems of classification and regression. For classification, we found that OCT improved significantly upon CART in synthetic and practical settings. OCT-H improves further and delivers performance similar to random forest and boosted trees. For regression problems, we found that ORT had small improvements over CART, and ORT-H further improved but did not reach the levels of random forests and boosting. We then adapted the optimal regression tree framework to allow the fitting of linear prediction models in the leaves, and the resulting methods ORT-L and ORT-LH achieved comparable performance to random forests and boosting. These results are significant as they permit the use of interpretable decision tree methods in practice without needing to sacrifice any performance compared to using random forests or boosting.

Next, we developed an approach for using optimal decision trees for the problem of prescription. This is a fundamentally different problem to prediction due to the absence of counterfactual data, and so we developed a procedure for imputing the missing observational data during the training of the prescriptive tree in order to obtain a tree that simultaneously minimizes prediction and prescription error. We showed on a variety of synthetic and practical examples that this approach leads to materially significant improvements in prescriptive performance compared to the existing state-of-the-art approaches in this area. In particular, we achieve competitive performances without sacrificing the interpretability of a single decision tree.

Finally, we presented a case study showing the impact that our improved methods can offer in a real-life practical setting. We applied OCT to the problem of diagnosing children with head trauma, a task that is currently performed in the US using a CART model, and so interpretability is crucial in this matter. We showed that our OCT approach was able to maintain the current levels of detection accuracy, whilst reducing the number of misdiagnoses and hence CT scans conducted by 25–50%. This is just one example of many showing how the performance improvements delivered by the Optimal Trees approach can translate to significant impact in the real world.

Bibliography

[1] TS Arthanari and Yadolah Dodge. *Mathematical Programming in Statistics*, volume 341. Wiley, New York, 1981.

[2] Shireen M Atabaki, Ian G Stiell, Jeffrey J Bazarian, Karin E Sadow, Tien T Vu, Mary A Camarca, Scott Berns, and James M Chamberlain. A clinical decision rule for cranial computed tomography in minor pediatric head trauma. *Archives of pediatrics & adolescent medicine*, 162(5):439–445, 2008.

[3] Susan Athey and Guido Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360, 2016.

[4] Peter Auer, Robert C Holte, and Wolfgang Maass. Theory and applications of agnostic pac-learning with small decision trees. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 21–29, 1995.

[5] Gabriel Baron, Elodie Perrodeau, Isabelle Boutron, and Philippe Ravaud. Reporting of analyses from randomized controlled trials with multiple arms: a systematic review. *BMC medicine*, 11(1):84, 2013.

[6] Hamsa Bastani and Mohsen Bayati. Online decision-making with high-dimensional covariates. *Available at SSRN 2661896*, 2015. Working paper.

[7] Kristin P Bennett. Decision tree construction via linear programming. In M. Evans, editor, *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference*, pages 97–101, 1992.

[8] Kristin P Bennett and J Blue. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214, 1996.

[9] Kristin P Bennett and Jennifer A Blue. A support vector machine approach to decision trees. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 3, pages 2396–2401. IEEE, 1998.

[10] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, pages 1–44, 2017.

[11] Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *arXiv preprint arXiv:1402.5481*, 2014.

[12] Dimitris Bertsimas and Nathan Kallus. Pricing from observational data. *arXiv preprint arXiv:1605.02347*, 2016.

[13] Dimitris Bertsimas and Angela King. An algorithmic approach to linear regression. *Operations Research, to appear*, 2015.

[14] Dimitris Bertsimas and Angela King. Logistic regression: From art to science. *Operations Research Center, MIT, submitted for publication*, 2015.

[15] Dimitris Bertsimas and Rahul Mazumder. Least quantile regression via modern optimization. *The Annals of Statistics*, 42(6):2494–2525, 2014.

[16] Dimitris Bertsimas and Romy Shioda. Classification and regression via integer optimization. *Operations Research*, 55(2):252–271, 2007.

[17] Dimitris Bertsimas and Robert Weismantel. *Optimization over Integers*. Dynamic Ideas, Belmont, Massachusetts, 2005.

[18] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *Annals of Statistics, to appear*, 2015.

[19] Dimitris Bertsimas, Nathan Kallus, Alex Weinstein, and Ying Daisy Zhuo. Personalized diabetes management using electronic medical records. *Diabetes Care*, 40(2):210–217, 2017.

[20] Dimitris Bertsimas, Colin Pawlowski, and Ying Zhuo. From predictive methods to missing data imputation: An optimization approach. *Journal of Machine Learning Research*, under review, 2017.

[21] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *arXiv preprint arXiv:1411.1607*, 2014.

[22] Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[23] Åke Björck. *Numerical methods for least squares problems.* SIAM, 1996.

[24] Kathy Boutis, William Cogollo, Jason Fischer, Stephen B Freedman, Guila Ben David, and Karen E Thomas. Parental knowledge of potential cancer risks from exposure to computed tomography. *Pediatrics*, 132(2):305–311, 2013.

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

[26] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[27] Leo Breiman. Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6):2350–2383, 1996.

[28] Leo Breiman. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.

[29] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[30] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.

[31] D Brenner, C Elliston, E Hall, and W Berdon. Estimated risks of radiation-induced fatal cancer from pediatric CT. *AJR. American journal of roentgenology*, 176(2):289–296, February 2001.

[32] David J Brenner. Estimating cancer risks from pediatric CT: going from the qualitative to the quantitative. *Pediatric radiology*, 32(4):228–1– discussion 242–4, April 2002.

[33] Dalia Buffery. The 2015 oncology drug pipeline: innovation drives the race to cure cancer. *American health & drug benefits*, 8(4):216, 2015.

[34] Probal Chaudhuri, Wen-Da Lo, Wei-Yin Loh, and Ching-Ching Yang. Generalized regression trees. *Statistica Sinica*, pages 641–666, 1995.

[35] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754*, 2016.

[36] International Warfarin Pharmacogenetics Consortium et al. Estimation of the warfarin dose with clinical and pharmacogenetic data. *N Engl J Med*, 2009 (360):753–764, 2009.

[37] Louis Anthony Cox Jr, QIU Yuping, and Warren Kuehner. Heuristic least-cost computation of discrete classification functions with uncertain argument values. *Annals of Operations Research*, 21(1):1–29, 1989.

[38] G. B. Dantzig. Programming of interdependent activities: II mathematical model. *Econometrica*, 17:200–211, 1949.

[39] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press and the RAND Corporation, 1963.

[40] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[41] J Dunning, J Patrick Daly, JP Lomas, F Lecky, J Batchelor, and K Mackway-Jones. Derivation of the children's head injury algorithm for the prediction of important clinical events decision rule for head injury in children. *Archives of disease in childhood*, 91(11):885–891, 2006.

[42] Joshua S Easter, Katherine Bakes, Jasmeet Dhaliwal, Michael Miller, Emily Caruso, and Jason S Haukoos. Comparison of PECARN, CATCH, and CHALICE rules for children with minor head injury: a prospective cohort study. *Annals of emergency medicine*, 64(2):145–52– 152.e1–5, August 2014.

[43] Saher Esmeir and Shaul Markovitch. Anytime learning of decision trees. *The Journal of Machine Learning Research*, 8:891–933, 2007.

[44] M Faul, L Xu, M M Wald, and V G Coronado. *Traumatic brain injury in the United States: Emergency Department Visits, Hospitalizations and Deaths 2002âĂŞ2006.* Centers for Disease Control and Prevention, National Center for Injury Prevention and Control, 2010.

[45] Michael L Feldstein, Edwin D Savlov, and Russell Hilf. A statistical model for predicting response of breast cancer patients to cytotoxic chemotherapy. *Cancer research*, 38(8):2544–2548, 1978.

[46] Patrick A Flume, Brian P O'sullivan, Karen A Robinson, Christopher H Goss, Peter J Mogayzel Jr, Donna Beth Willey-Courand, Janet Bujan, Jonathan Finder, Mary Lester, Lynne Quittell, et al. Cystic fibrosis pulmonary guidelines: chronic medications for maintenance of lung health. *American journal of respiratory and critical care medicine*, 176(10):957–969, 2007.

[47] National Collaborating Centre for Acute Care (UK et al. Head injury: triage, assessment, investigation and early management of head injury in infants, children and adults. 2007.

[48] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[49] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[50] Salvador Garcia and Francisco Herrera. An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of Machine Learning Research*, 9(Dec):2677–2694, 2008.

[51] M. Garey and D. Johnson. *A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[52] John C Gittins. *Multi-Armed Bandit Allocation Indices*. Wiley, Chichester, UK, 1989.

[53] Alexander Goldenshluger and Assaf Zeevi. A linear response bandit problem. *Stochastic Systems*, 3(1):230–261, 2013.

[54] Marjan Gort, Manda Broekhuis, Renée Otter, and Niek S Klazinga. Improvement of best practice in early breast cancer: actionable surgeon and hospital factors. *Breast cancer research and treatment*, 102(2):219–226, 2007.

[55] Thomas Grubinger, Achim Zeileis, and Karl-Peter Pfeiffer. evtree: Evolutionary learning of globally optimal classification and regression trees in r. *Journal of statistical software*, 61(1):1–29, 2014. ISSN 1548-7660. doi: 10.18637/jss.v061.i01. URL https://www.jstatsoft.org/v061/i01.

[56] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015. http://www.gurobi.com.

[57] Gurobi Optimization Inc. Gurobi 7.0 performance benchmarks. http://www.gurobi.com/pdfs/benchmarks.pdf, 2016. Accessed 17 December 2016.

[58] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *IJCAI*, pages 1002–1007. Citeseer, 1993.

[59] David George Heath. *A geometric framework for machine learning.* PhD thesis, Johns Hopkins University, 1993.

[60] Harold V Henderson and Paul F Velleman. Building multiple regression models interactively. *Biometrics*, pages 391–411, 1981.

[61] Jennifer L Hill. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, 2011.

[62] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[63] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674, 2006.

[64] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[65] IBM ILOG CPLEX. V12.1 userâĂŹs manual, 2014.

[66] Kentaro Ide, Satoko Uematsu, Kenichi Tetsuhara, Satoshi Yoshimura, Takahiro Kato, and Tohru Kobayashi. External validation of the PECARN head trauma prediction rules in japan. *Academic Emergency Medicine*, 2016.

[67] Kosuke Imai and Marc Ratkovic. Covariate balancing propensity score. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):243–263, 2014.

[68] Thomas R Insel. Translating scientific opportunity into public health impact: a strategic plan for research on mental illness. *Archives of General Psychiatry*, 66(2):128–133, 2009.

[69] Amir Jaffer and Lee Bragg. Practical tips for warfarin dosing and monitoring. *Cleveland Clinic journal of medicine*, 70(4):361–371, 2003.

[70] Nathan Kallus. Recursive partitioning for personalization using observational data. In *International Conference on Machine Learning*, pages 1789–1798, 2017.

[71] Anna Karpas, Marsha Finkelstein, and Samuel Reid. Which management strategy do parents prefer for their head-injured child: immediate computed tomography scan or observation? *Pediatric emergency care*, 29(1):30–35, 2013.

[72] Hyunjoong Kim, Wei-Yin Loh, Yu-Shan Shih, and Probal Chaudhuri. Visualizable and interpretable regression models with good prediction power. *IIE Transactions*, 39(6):565–579, 2007.

[73] Nathan Kuppermann, James F Holmes, Peter S Dayan, John D Hoyle, Shireen M Atabaki, Richard Holubkov, Frances M Nadel, David Monroe, Rachel M Stanley, Dominic A Borgialli, et al. Identification of children at very low risk of clinically-important brain injuries after head trauma: a prospective cohort study. *The Lancet*, 374(9696):1160–1170, 2009.

[74] Robert J LaLonde. Evaluating the econometric evaluations of training programs with experimental data. *The American economic review*, pages 604–620, 1986.

[75] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[76] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002. URL http://CRAN.R-project.org/doc/Rnews/.

[77] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[78] Dong-Joon Lim, Shabnam R Jahromi, Timothy R Anderson, and Anca-Alexandra Tudorie. Comparing technological advancement of hybrid electric vehicles (hev) in different market segments. *Technological Forecasting and Social Change*, 97:140–153, 2015.

[79] Ilya Lipkovich and Alex Dmitrienko. Strategies for identifying predictive biomarkers and subgroups with enhanced treatment effect in clinical trials using sides. *Journal of biopharmaceutical statistics*, 24(1):130–153, 2014.

[80] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data.* John Wiley & Sons, 2014.

[81] Wei-Yin Loh. Regression tress with unbiased variable selection and interaction detection. *Statistica Sinica*, pages 361–386, 2002.

[82] Wei-Yin Loh and Yu-Shan Shih. Split selection methods for classification trees. *Statistica sinica*, pages 815–840, 1997.

[83] Asdrúbal López-Chau, Jair Cervantes, Lourdes López-García, and Farid García Lamont. FisherâĂŹs decision tree. *Expert Systems with Applications*, 40(16): 6283–6291, 2013.

[84] F Lorton, C Poullaouec, E Legallais, J Simon-Pimmel, M A Chêne, H Leroy, M Roy, E Launay, and C Gras-Le Guen. Validation of the PECARN clinical decision rule for children with minor head trauma: a French multicenter prospective study. *Scandinavian journal of trauma, resuscitation and emergency medicine*, 24(1):98, August 2016.

[85] Jennifer R Marin, Matthew D Weaver, Amber E Barnato, Jonathan G Yabes, Donald M Yealy, and Mark S Roberts. Variation in emergency department head computed tomography use for pediatric head trauma. *Academic emergency medicine : official journal of the Society for Academic Emergency Medicine*, 21 (9):987–995, September 2014.

[86] Jennifer R Marin, Matthew D Weaver, Donald M Yealy, and Rebekah C Mannix. Trends in visits for traumatic brain injury to emergency departments in the United States. *JAMA : the journal of the American Medical Association*, 311(18):1917–1919, May 2014.

[87] Diana L Miglioretti, Eric Johnson, Andrew Williams, Robert T Greenlee, Sheila Weinmann, Leif I Solberg, Heather Spencer Feigelson, Douglas Roblin, Michael J Flynn, Nicholas Vanneman, and Rebecca Smith-Bindman. The use of computed tomography in pediatrics and the associated radiation exposure and estimated cancer risk. *JAMA pediatrics*, 167(8):700–707, August 2013.

[88] William R Mower, Jerome R Hoffman, Mel Herbert, Allan B Wolfson, Charles V Pollack Jr, Michael I Zucker, NEXUS II Investigators, et al. Developing a decision instrument to guide computed tomographic imaging of blunt head injury patients. *Journal of Trauma and Acute Care Surgery*, 59(4):954–959, 2005.

[89] Sreerama Murthy and Steven Salzberg. Lookahead and pathology in decision tree induction. In *IJCAI*, pages 1025–1033. Citeseer, 1995.

[90] Sreerama K Murthy and Steven Salzberg. Decision tree induction: How effective is the greedy heuristic? In *KDD*, pages 222–227, 1995.

[91] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[92] George L Nemhauser. Integer Programming: the Global Impact. Presented at EURO, INFORMS, Rome, Italy, 2013. http://euro-informs2013.org/data/http_/euro2013.org/wp-content/uploads/nemhauser.pdf, 2013. Accessed 9 September 2015.

[93] Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, pages 1720–1728, 2015.

[94] Steven W Norton. Generating better decision trees. In *IJCAI*, volume 89, pages 800–805, 1989.

[95] Martin H Osmond, Terry P Klassen, George A Wells, Rhonda Correll, Anna Jarvis, Gary Joubert, Benoit Bailey, Laurel Chauvin-Kimoff, Martin Pusic, Don McConnell, et al. Catch: a clinical decision rule for the use of computed tomography in children with minor head injury. *Canadian Medical Association Journal*, 182(4):341–348, 2010.

[96] Michael J Palchak, James F Holmes, Cheryl W Vance, Rebecca E Gelber, Bobbie A Schauer, Mathew J Harrison, Jason Willis-Shore, Sandra L Wootton-Gorges, Robert W Derlet, and Nathan Kuppermann. A decision rule for identifying children at low risk for brain injuries after blunt head trauma. *Annals of emergency medicine*, 42(4):492–506, 2003.

[97] A Pandor, S Goodacre, S Harnan, M Holmes, A Pickering, P Fitzgerald, A Rees, and M Stevenson. Diagnostic management strategies for adults and children with minor head injury: a systematic review and an economic evaluation. 2011.

[98] Mahesh KB Parmar, James Carpenter, and Matthew R Sydes. More multiarm randomised trials of superiority are needed. *The Lancet*, 384(9940):283, 2014.

[99] Harold J Payne and William S Meisel. An algorithm for constructing optimal binary decision trees. *Computers, IEEE Transactions on*, 100(9):905–916, 1977.

[100] Alastair Pickering, Susan Harnan, Patrick Fitzgerald, Abdullah Pandor, and Steve Goodacre. Clinical decision rules for children with minor head injury: a systematic review. *Archives of Disease in Childhood*, 96(5):414–421, May 2011.

[101] Scott Powers, Junyang Qian, Kenneth Jung, Alejandro Schuler, H. Shah, Nigam, Trevor Hastie, and Robert Tibshirani. Some methods for heterogenous treatment effect estimation in high dimensions. *arXiv preprint arXiv:1707.00102v1*, 2017. Working paper.

[102] Min Qian and Susan A Murphy. Performance guarantees for individualized treatment rules. *Annals of statistics*, 39(2):1180, 2011.

[103] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[104] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[105] John R Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. Singapore, 1992.

[106] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015. URL http://www.R-project.org/.

[107] Paul R Rosenbaum and Donald B Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, pages 41–55, 1983.

[108] Maura E Ryan, Susan Palasis, Gaurav Saigal, Adam D Singer, Boaz Karmazyn, Molly E Dempsey, Jonathan R Dillman, Christopher E Dory, Matthew Garber, Laura L Hayes, et al. Acr appropriateness criteria head traumaâĂŤchild. *Journal of the American College of Radiology*, 11(10):939–947, 2014.

[109] Nguyen Hung Son. From optimal hyperplanes to optimal decision trees. *Fundamenta Informaticae*, 34(1, 2):145–174, 1998.

[110] Terry Therneau, Beth Atkinson, and Brian Ripley. *rpart: Recursive Partitioning and Regression Trees*, 2015. URL http://CRAN.R-project.org/package=rpart. R package version 4.1-9.

[111] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[112] Christos Tjortjis and John Keane. T3: a classification algorithm for data mining. In *Lecture Notes in Computer Science*, volume 2412, pages 50–55. Springer, 2002.

[113] Top500 Supercomputer Sites. Performance development. http://www.top500.org/statistics/perfdevel/, 2016. Accessed 17 December 2016.

[114] Luis Torgo. Regression data sets. http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html, 2017. Accessed 9 November 2017.

[115] Luís Fernando Raínho Alves Torgo. *Inductive learning of tree-based regression models*. PhD thesis, Universidade do Porto, 1999.

[116] Alfred Truong. *Fast growing and interpretable oblique trees via logistic regression models*. PhD thesis, University of Oxford, 2009.

[117] Panagiotis Tzirakis and Christos Tjortjis. T3c: improving a decision tree classification algorithmâĂŹs interval splits on continuous attributes. *Advances in Data Analysis and Classification*, pages 1–18, 2016.

[118] Stefan Wager and Susan Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, (to appear), 2017.

[119] Daniel Westreich, Justin Lessler, and Michele Jonsson Funk. Propensity score estimation: machine learning and classification methods as alternatives to logistic regression. *Journal of clinical epidemiology*, 63(8):826, 2010.

[120] DC Wickramarachchi, BL Robertson, M Reale, CJ Price, and J Brown. Hhcart: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.

[121] Larry Winner. Miscellaneous data sets. http://www.stat.ufl.edu/~winner/datasets.html, 2017. Accessed 9 November 2017.

[122] Roger Zemek, S Duval, C Dematteo, B Solomon, M Keightley, M Osmond, et al. Guidelines for diagnosing and managing pediatric concussion. *Ontario Neurotrauma Foundation*, 2014.

[123] Xin Zhou, Nicole Mayer-Hamblett, Umer Khan, and Michael R Kosorok. Residual weighted learning for estimating individualized treatment rules. *Journal of the American Statistical Association*, 112(517):169–187, 2017.