

*Machine Learning under
a Modern Optimization Lens*

Dimitris Bertsimas

Jack Dunn

Operations Research Center
Massachusetts Institute of Technology

Preliminary version

September 5, 2018

Do not circulate without
permission from the authors

Dynamic Ideas LLC

Belmont, Massachusetts

Contents

Preface	x
1 The Optimization Lenses	1
1.1 The Optimization Lenses	2
1.2 The Remarkable Progress of MIO	4
1.3 What is Tractable?	5
I Regression and Extensions	7
2 Robust Regression	9
2.1 Norms and their Duals	11
2.2 Equivalence of Robustification and Regularization	13
2.3 Non-equivalence of robustification and regularization	16
2.4 The Edge of Robust Regression	21
2.5 Concluding Remarks	21
2.6 Notes and Sources	22
3 Sparse Regression	25
3.1 A Primal Approach to Sparse Linear Regression	26
3.2 Computational Insights from the Primal Algorithm	30
3.3 A Dual Approach to Sparse Linear Regression	39
3.4 A cutting plane algorithm	42
3.5 Scalability and Phase Transitions	45
3.6 Concluding Remarks	53
3.7 Notes and Sources	53
4 Nonlinear Regression	55
4.1 Convex Regression	56
4.2 Sparse Convex Regression	59
4.3 Median Regression	61
4.4 Concluding Remarks	66
4.5 Notes and Sources	66

5 Holistic Regression	67
5.1 Desirable Properties of a Linear Regression Model	68
5.2 Low Global Multicollinearity	72
5.3 Examples of Holistic Regression	76
5.4 Insights from Computations	80
5.5 Concluding Remarks	91
5.6 Notes and Sources	92
6 Sparse Classification	93
6.1 Regularized classification	94
6.2 The dual method to sparse classification	97
6.3 Sparse logistic regression	99
6.4 Sparse Support Vector Machine	105
6.5 Concluding Remarks	107
6.6 Notes and Sources	108
II Optimal Trees for Classification and Regression 111	
7 Classification and Regression Trees	113
7.1 Overview of CART	114
7.2 Limitations of CART	118
7.3 Random Forests and Boosted Trees	121
7.4 Notes and Sources	122
8 Optimal Classification Trees with Parallel Splits	125
8.1 Review of Classification Tree Methods	126
8.2 A Mixed-Integer Optimization Approach	130
8.3 A Local Search Approach	137
8.4 A Method for Tuning Hyperparameters	145
8.5 Experiments with Synthetic Datasets	154
8.6 Experiments with Real-World Datasets	160
8.7 Concluding Remarks	167
8.8 Notes and Sources	168
9 Optimal Classification Trees with Hyperplane Splits	169
9.1 OCT with Hyperplanes via MIO	171
9.2 OCT with Hyperplanes via Local Search	174
9.3 Interpretability of Parallel and Hyperplane Trees	178
9.4 Experiments with Synthetic Datasets	180
9.5 Experiments with Real-World Datasets	188
9.6 Concluding Remarks	192
9.7 Notes and Sources	192
10 Optimal Regression Trees with Constant Predictions	195
10.1 Review of Regression Tree Methods	196

10.2 ORT with Constant Predictions via MIO	198
10.3 ORT with Constant Predictions via Local Search	200
10.4 Experiments with Synthetic Datasets	203
10.5 Experiments with Real-World Datasets	213
10.6 Concluding Remarks	219
10.7 Notes and Sources	220
11 Optimal Regression Trees with Linear Predictions	221
11.1 ORT with Linear Predictions via MIO	223
11.2 ORT with Linear Predictions via Local Search	225
11.3 Experiments with Synthetic Datasets	228
11.4 Experiments with Real-World Datasets	233
11.5 Concluding Remarks	237
11.6 Notes and Sources	237
12 Optimal Trees and Neural Networks	239
12.1 Mathematical Formulation of Neural Networks	241
12.2 FNNs and Optimal Trees	247
12.3 CNNs and Optimal Trees	254
12.4 RNNs and Optimal Trees	261
12.5 Transforming OCT-Hs into Classification NNs	270
12.6 Computational Results	275
12.7 Concluding Remarks	276
12.8 Notes and Sources	276
III Prescriptive Analytics	281
13 From Predictive to Prescriptive Analytics	283
14 Optimal Prescription Trees	285
14.1 Optimal Prescriptive Trees	291
14.2 Experiments with Synthetic Datasets	295
14.3 Experiments with Real-World Datasets	302
14.4 Concluding Remarks	310
14.5 Notes and Sources	310
IV Unsupervised Methods	311
15 Optimal Missing Data Imputations	313
15.1 Methods for Optimal Imputation	315
15.2 K -NN Based Imputation	317
15.3 Mixed SVM Based Imputation	322
15.4 Tree Based Imputation	325
15.5 Model Selection	327

15.6 Real-World Data Experiments	329
15.7 Concluding Remarks	344
15.8 Notes and Sources	345
16 The Power of Optimization Over Randomization	349
16.1 Limitations of Randomization	351
16.2 Optimization Approach	353
16.3 Optimization for Reducing Discrepancies	356
16.4 Optimization, Randomization, and Bias	359
16.5 Optimization for Reaching a Conclusion	361
16.6 Concluding Remarks	363
16.7 Notes and Sources	364
17 Identifying Exceptional Responders	365
17.1 Identifying interpretable optimal subgroups	367
17.2 Computational experiments with synthetic data sets	373
17.3 Computational experiments with real-world datasets	377
17.4 Concluding Remarks	380
17.5 Notes and Sources	381
18 Interpretable Clustering	383
19 The Bootstrap	385
V Matrix Methods	387
20 Sparse Principal Component Analysis	389
21 Factor Analysis	391
22 Sparse Inverse Covariance Estimation	393
23 Matrix Completion	395
24 Learning from Tensors	397
VI Machine Learning in Action	399
25 Diagnostics for Children After Head Trauma	401
25.1 Data and Methods	402
25.2 Results	405
25.3 Discussion	411
25.4 Notes and Sources	413
26 Prediction of mortality for liver transplant allocation	415

<i>Contents</i>	ix
26.1 Data and Methods	416
26.2 Results	421
26.3 Discussion	424
26.4 Notes and Sources	427
27 The Final Word	429
References	431
Index	453

Preface

The goal is to turn data into information, and information into insight.
Carly Fiorina

The purpose of this book is to provide a unified, insightful, and modern treatment of machine learning using three modern optimization lenses: convex, robust and mixed integer optimization.

Machine learning has experienced tremendous growth in the twenty-first century that has influenced society in a variety of areas of human activity. The majority of the central problems of machine learning have been addressed using heuristic methods even though they can be formulated as formal optimization problems. Examples include Lasso for sparse regression, classification and regression trees (CART) for classification and regression among many others. Since the early 1990s, optimization theory and especially convex, robust and mixed integer optimization (MIO) has made significant advances in our ability to model and solve high dimensional problems. As an example, algorithmic advances in MIO coupled with hardware improvements have resulted in an astonishing 2 trillion factor speedup in solving MIO problems since the early 1990s.

Our overarching goal in this book is to revisit the central problems of machine learning using formal optimization methods and demonstrate that they can greatly benefit from a modern optimization treatment. We take a rigorous, non-heuristic, optimization-based approach to machine learning that leads to better out-of-sample performance compared to heuristic approaches. Specifically, throughout the book we demonstrate that using modern optimization we can find solutions to large scale instances of central problems in machine learning that

- (a) can be found in seconds/minutes.
- (b) can be certified to be optimal in minutes/hours.
- (c) outperform classical heuristic approaches in out-of-sample experiments involving real-world and synthetic datasets.

We give special attention to theory, but also cover applications and present case studies. Our main objective is to help the reader become a sophisticated practitioner or researcher of machine learning. More specifically, we wish to develop the ability to formulate machine learning problems as optimization problems, provide an appreciation of the main

classes of problems that are practically solvable, describe the available solution methods, and build an understanding of the qualitative properties of the solutions they provide.

Our general philosophy is that insight matters most. For the subject matter of this book, this necessarily requires a computational approach. Another of our objectives is to be comprehensive, but economical. We have made an effort to cover and highlight all of the principal ideas in this approach to machine learning. However, we have not tried to be encyclopedic, or to discuss every possible detail relevant to a particular algorithm. Our premise is that once mature understanding of the basic principles is in place, further details can be acquired by the reader with little additional effort.

Our last objective is to bring the reader up to date with respect to the state of the art. The totality of the material we present in this book is from our research, either individually or jointly, that has very recently appeared or is appearing in the academic literature.

The success of the approach outlined in this book hinges on its ability to deal with large and important problems. In that sense, Part VI of the book, “Machine Learning in Action”, is a critical piece of this book. It will, we hope, convince the reader that progress on challenging problems requires both problem specific insight, as well as a deeper understanding of the underlying theory.

Structure of the Book

The book is organized into six parts:

Part I consisting of Chapters 2–6 describes regression and its extensions.

Part II consisting of Chapters 7–12 contains optimal classification and regression trees.

Part III consisting of Chapters 13–14 introduces prescriptive methods in machine learning.

Part IV consisting of Chapters 15–19 describes unsupervised methods for a variety of machine learning problems.

Part V consisting of Chapters 20–24 develops matrix methods.

Part VI consisting of Chapters 25–26 details the application of the methods introduced in earlier chapters in the real-world.

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

Chapter 1: Introduces the optimization lenses we use in this book: convex, robust and mixed integer optimization; describes the astonishing progress of mixed integer optimization and discusses what is tractable.

Chapter 2: Develops robust linear regression under the lens of robust optimization, characterizes precisely its relationship with regularized regression and suggests that the remarkable success **Lasso** has experienced

since the mid-1990s can be attributed to its robustness rather than its sparsity properties.

Chapter 3: Proposes both primal and dual methods to solve sparse linear regression under the lens of mixed integer and convex integer optimization, solves sparse linear regression in dimensions and samples in the 100,000s in seconds, observes new phase transition phenomena and argues that the dual sparse regression approach presents a superior alternative over heuristic methods available at present.

Chapter 4: Contains extensions to nonlinear and median regression under the lens of mixed integer and convex optimization.

Chapter 5: Outlines holistic regression, a framework based on mixed integer optimization, which simultaneously develops a linear regression with a variety desirable properties such as robustness, sparsity, significance, absence of multi-collinearity, among other properties for linear regression.

Chapter 6: Generalizes robustness and sparsity to logistic regression and provides a framework for holistic logistic regression.

Chapter 7: Gives an overview of the classification and regression trees (CART) algorithm, random forests and gradient boosted decision trees, and outlines some of their limitations.

Chapter 8: 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 9: 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 10: 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 11: 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 12: Proves that a variety of neural networks (feedforward, convolutional and recurrent) can be transformed to classification and regression trees with hyperplanes with the same accuracy in the training set, showing that such trees are at least as powerful as neural networks in modeling power.

Chapter 13: Proposes a framework for extending predictive machine learning methods to prescriptive ones, and demonstrates that such methods provide an edge in decision making directly from data.

Chapter 14: 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 15: Poses the missing data problem under a general optimization framework and develops `opt.impute`, an algorithm for missing data imputation that significantly outperforms other heuristic approaches.

Chapter 16: Provides theoretical and computational evidence that groups created by optimization have exponentially lower discrepancy in pre-treatment covariates than those created by randomization or by existing matching methods.

Chapter 17: Identifies a subgroup in a clinical trial for which the average treatment effect is exceptionally strong or exceptionally weak and which can be defined by a small pre-specified number of covariates under the lens of mixed integer optimization.

Chapter 18: Proposes a new method for clustering that is interpretable and provides insights on the nature of the clusters by extending the methodology from optimal classification trees.

Chapter 19: Takes a different perspective on the bootstrap, one of the most significant ideas of modern statistics. The bootstrap uses randomization, but in this chapter we use exact counting of integer points in polyhedra to propose a method that has an edge on accuracy compared to randomization.

Chapter 20: Develops an approach to sparse principal component analysis using mixed integer optimization and demonstrates that it has an edge over alternative heuristic methods.

Chapter 21: Provides a rigorous framework for factor analysis under the lenses of convex and mixed integer optimization that leads to provably optimal solutions in high dimensions.

Chapter 22: Extends the framework for sparse regression developed from earlier chapters to sparse inverse covariance estimation and demonstrates its edge over heuristic approaches.

Chapter 23: Introduces robustness and sparsity for estimation problems in matrices and develops new algorithms for matrix completion using ideas from all of the optimization lenses in this book.

Chapter 24: Generalizes the ideas of Chapter 23 to tensors and includes examples that demonstrate the edge of these methods compared to other methods.

Chapter 25: Applies optimal classification trees to detecting critical brain injury in children and demonstrates that the edge in prediction accuracy leads to a decrease in CT-scans and a reduction in missed diagnoses.

Chapter 26: Details the development of optimal classification trees in redesigning the system of liver transplantation in the United States that

promises to avert 400 deaths annually.

Chapter 27: Summarizes the key messages of the book and includes some closing thoughts.

Philosophy

Some of the key philosophical principles that characterize the book are:

- (a) **Interpretability.** We believe that interpretability in machine learning matters. Especially in critical applications involving decisions of significant magnitude, it has been our experience that decision makers need to understand the logic of the algorithm. We have placed particular emphasis on the ideas of sparsity that lead to interpretable regression models and to the development of optimal trees that are treated extensively in Part II of this book as well as in prescriptive trees, Chapter 14, and interpretable clustering, Chapter 18.
- (b) **Machine learning and optimization.** Historically statistics has been linked to probability theory. One of our objectives is to reveal that the link of machine learning/statistics to optimization leads to significant advances in our ability to solve machine learning/statistics problems and to provide a fresh perspective that enhances our understanding of machine learning/statistics.
- (c) **The importance of data driven approaches.** In our view, the only objective reality is data. Models exist primarily in our imagination and only provide approximations of reality. The entire approach in this book starts with data and then applies optimization to compute in large scale.
- (d) **Randomization versus optimization.** In several chapters in the book (Chapters 16, 17 and 19) we show that the use of optimization versus randomization methods leads to significant performance advances.
- (e) **Practability.** In contrast to complexity theory, we judge methods based on their ability to solve instances in times and for sizes that are appropriate for the application that motivated the problem. In our view, polynomial solvability or *NP*-hardness of a problem does not give relevant information for our ability to solve the problem in the real world. Given that the motivation of this book is to solve real world problems, we will use the notion of practical tractability (practability) alongside theoretical tractability when evaluating our algorithms.
- (f) **Prescriptive methods.** The majority of machine learning has focused on prediction. It is our belief that the ultimate objective

should be the ability to make high quality decisions. For this reason we have devoted Part III of the book to prescriptive methods in machine learning.

Acknowledgements

Part II, Chapter 14 and Part VI are from the joint work of the authors, while Part I, Chapter 13, Parts IV and V is from the joint work of Dimitris with his students, postdocs and colleagues. Specifically, Dimitris would like to express his appreciation to his co-authors:

Lauren Berk for her contributions in Chapter 20.

Martin Copenhaver for his contributions in Chapters 2 and 21.

Mac Johnson for his contribution in Chapter 16.

Nathan Kallus for his contributions in Chapters 13 and 16.

Angela King for her contributions in Chapters 3, 5 and 6.

Jourdain Lamperski for his contributions in Chapter 22.

Michael Li for his contributions in Chapters 5 and 23.

Rahul Mazumder for his contributions in Chapters 3, 12, 21 and 23.

Nishanth Mundru for his contributions in Chapters 4 and 14.

Agni Orfanoudaki and Holly Wiberg for their contributions in Chapter 18.

Jean Pauphilet for his contributions in Chapters 6 and 22.

Bart van Parys for his contributions in Chapters 3, 4 and 6.

Colin Pawlowski for his contributions in Chapters 6, 15 and 24.

Matthew Sobiesk for his contributions in Chapter 12.

Brad Sturt for his contributions in Chapter 19.

Nikos Trichakis for his contributions in Chapter 26.

Yuchen Wang for his contributions in Chapters 25 and 26.

Alex Weinstein and Nikita Korolko for their contributions in Chapter 17.

Daisy Zhuo for her contributions in Chapters 6 and 15.

Most importantly, we are grateful to our partners Georgia and Jen for their love and support in the course of this project.

*Dimitris Bertsimas
Jack Dunn
Cambridge, January 2019*

Chapter 1

The Optimization Lenses

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

– George Dantzig

Contents

- 1.1. The Optimization Lenses
- 1.2. The Remarkable Progress of MIO
- 1.3. What is Tractable?

The modern era of optimization started with George Dantzig, who in 1947 invented the simplex method to solve linear optimization (LO) problems. The earliest problems were addressed by the National Bureau of Standards and the RAND Corporation, and could solve a variety of problems with as many as 45 constraints and 70 variables in about 8 hours. Fast forward to the late 2010s, where currently we can solve MIOs with millions of variables and constraints in seconds/minutes using commercially available solvers.

In the mid 1980s, Narendra Karmarkar initiated an approach to LO called interior point methods. In the late 1980s, Yuri Nesterov and Arkadi Nemirovski began the application of interior point algorithms to conic optimization problems such as semidefinite and second order problems. Today we can solve semidefinite optimization problems in dimension in the thousands and second order cone problems with dimensions in the millions.

Robust optimization started in the late 1990s with the work of Arkadi Nemirovski and Ronny Ben Tal for convex optimization problems, and in the early 2000s with the work of Dimitris Bertsimas and Melynn Sim for discrete optimization problems. Today robust MIO problems with millions of variables and constraints can be solved in seconds/minutes.

In this chapter, we introduce the three optimization lenses we bring into play to address central problems in machine learning: convex, robust and mixed integer optimization. We further discuss the remarkable progress of MIO which is the central methodology we use in the book, and discuss our perspective on what is tractable.

1.1 The Optimization Lenses

Mixed Integer Optimization

The first optimization lens we use in this book is the mixed integer optimization model:

$$\begin{aligned} (\text{MIO}) \quad & \max \quad \mathbf{c}^T \mathbf{x} + \mathbf{h}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{Ax} + \mathbf{By} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^n \\ & \mathbf{y} \in \mathbb{R}_+^m, \end{aligned}$$

and the mixed integer quadratic integer optimization model:

$$\begin{aligned} (\text{QMIO}) \quad & \max \quad \mathbf{x}^T \mathbf{Qx} + \mathbf{c}^T \mathbf{x} + \mathbf{h}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{Ax} + \mathbf{By} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}_+^n \\ & \mathbf{y} \in \mathbb{R}_+^m. \end{aligned}$$

Since the early 1950s, the field of Operations Research has realized that MIO has exceptional modeling power and in the early years (1950s-

1980s) this modeling power came at a significant price, as at that time MIO could not scale to large scale problems. As we outline in Section 1.2, algorithmic advances in MIO coupled with hardware improvements have resulted in an astonishing over 2 trillion factor speedup in solving MIO problems since the early 1990s. Computational approaches to central problems in machine learning have started to being formulated in the 1970s-1990s, and during this period MIO was only capable of solving small scale instances. Correspondingly the belief was formed that MIO can not solve large scale ML problems. However, such beliefs have not been updated despite the astonishing advances in MIO.

Convex Optimization

The second lens we use is convex optimization:

$$\begin{aligned} (\text{CO}) \quad & \min f(\mathbf{x}) \\ \text{s.t.} \quad & g_j(\mathbf{x}) \leq 0, \quad j = 1, \dots, k, \end{aligned}$$

where the functions $f(\mathbf{x})$, $g_j(\mathbf{x})$, $j = 1, \dots, k$ are convex. One of the most important developments in the late 1980s-1990s was the development of interior point methods and since the 2000s first order methods (gradient descent and stochastic gradient descent) that scale in very high dimensions and have fast running times.

From the late 1980s to present, the most important developments were semi-definite optimization

$$\begin{aligned} (\text{SDO}) \quad & \min \mathbf{C} \bullet \mathbf{X} \\ \text{s.t.} \quad & \mathbf{A}_i \bullet \mathbf{X} \geq b_i, \quad i = 1, \dots, m \\ & \mathbf{X} \succeq \mathbf{0}, \end{aligned}$$

where $\mathbf{C} \bullet \mathbf{X} = \sum_{i,j=1}^n c_{ij}x_{ij}$ is element-wise multiplication and $\mathbf{X} \succeq \mathbf{0}$ means $\mathbf{u}^T \mathbf{X} \mathbf{u} \geq 0$ for all \mathbf{u} . Semidefinite optimization has also made significant progress since the 1980s and it scales to problems with \mathbf{X} with dimensions in the 1000s. When \mathbf{X} is diagonal, this leads to linear optimization which scales to problems in million of dimensions.

Robust Optimization

The third lens we use is robust optimization:

$$\begin{aligned} (\text{RO}) \quad & \min \max_{\mathbf{c} \in \mathcal{U}} \mathbf{c}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{a}_i^T \mathbf{x} \geq b_i, \quad \forall \mathbf{a}_i \in \mathcal{U}_i, \quad i = 1, \dots, m, \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

where \mathcal{U} , \mathcal{U}_i , $i = 1, \dots, m$ are uncertainty sets that model the uncertainty in the data. Since the late 1990s, robust optimization has been established as a tractable way to solve optimization problems under uncertainty as for many optimization problems the time to solve the robust optimization problem is of the same order of magnitude as the nominal problem.

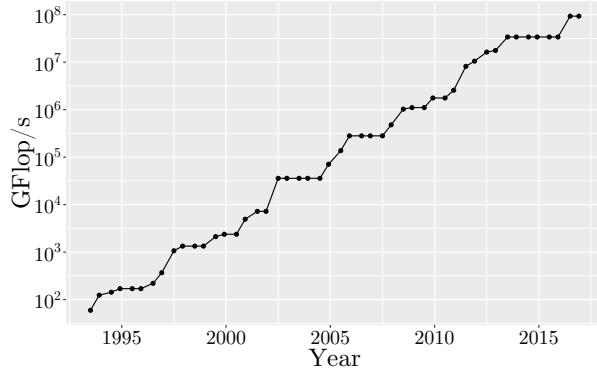
1.2 The Remarkable Progress of MIO

Continuous optimization methods have been widely used in statistics since the 1960s, but MIO methods, which have been used to great effect in many other fields, have not had the same impact in ML/statistics. Despite the knowledge that many ML problems have natural MIO formulations [2], there is the belief within ML/statistics community that MIO problems are intractable even for small to medium instances, which was true in the 1950s-1980s when the first continuous optimization methods for ML/statistics were being developed.

However, since the early 1990s we have seen an incredible increase in the computational power of MIO solvers, and modern MIO solvers such as GUROBI [125] and CPLEX [147] are able to solve linear MIO problems of considerable size. To quantify this increase, Bixby [46] 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 [46, 203]. 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 [46, 203, 126], 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 [46]. Coupled with the increase in computer hardware during this same period as shown in Figure 1.1, a factor of approximately 1,560,000 [254], 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 [37, 30, 31, 33].

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 longer supported. In this book, 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.

Figure 1.1: Peak supercomputer speed in GFlop/s (log scale) from 1994–2016.



1.3 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 [109], 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 it 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 [77, 78] 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 book is to solve real world problems, we will use the notion of practical tractability alongside theoretical tractability when evaluating our algorithms.

Notation

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

Part I

Regression and Extensions

Chapter 2

Robust Regression

Robustness is more important than optimality

– Anonymous

Contents

- 2.1. Norms and their Duals
- 2.2. Equivalence of Robustification and Regularization
- 2.3. Non-equivalence of robustification and regularization
- 2.4. The Edge of Robust Regression
- 2.5. Concluding Remarks
- 2.6. Notes and Sources

Given input data $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathbb{R}^{n \times p}$ and response data $\mathbf{y} = (y_1, \dots, y_n) \in \mathbb{R}^n$, the linear regression problem is as follows:

$$\min_{\boldsymbol{\beta}} g(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}), \quad (2.1)$$

where $g(\mathbf{a}) = \|\mathbf{a}\|_2$ corresponds to the classical least squares problem, while $g = \|\mathbf{a}\|_1$ is known as least absolute deviation (LAD). In favor of models which mitigate the effects of overfitting these are often replaced by the *regularization* problem

$$\min_{\boldsymbol{\beta}} g(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + h(\boldsymbol{\beta}),$$

where $h : \mathbb{R}^n \rightarrow \mathbb{R}$ is some penalty function, typically taken to be convex. This approach often aims to address overfitting by penalizing the complexity of the model, measured as $h(\boldsymbol{\beta})$. For example, taking $g = \ell_2^2$ and $h = \ell_2^2$, we recover the so-called regularized least squares (RLS), also known as ridge regression [105]. The choice of $g = \ell_2^2$ and $h = \ell_1$ leads to **Lasso**, or least absolute shrinkage and selection operator, introduced in [251]. **Lasso** is often employed in scenarios where the solution $\boldsymbol{\beta}$ is desired to be sparse, i.e., $\boldsymbol{\beta}$ has very few nonzero entries.

In contrast to this approach, one may alternatively wish to re-examine the nominal regression problem $\min_{\boldsymbol{\beta}} g(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$ and instead attempt to solve this taking into account adversarial noise in the data matrix \mathbf{X} . As in [111, 12], this approach may take the form

$$\min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}} g(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}), \quad (2.2)$$

where the set $\mathcal{U} \subseteq \mathbb{R}^{n \times p}$ characterizes the user's belief about uncertainty on the data matrix \mathbf{X} . This set \mathcal{U} is known in the language of robust optimization [12, 20] as an uncertainty set and the inner maximization problem $\max_{\Delta \in \mathcal{U}} g(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta})$ takes into account the worst-case error (measured via g) over \mathcal{U} . We call such a procedure *robustification* because it attempts to immunize or robustify the regression problem from structural uncertainty in the data. Such an adversarial or "worst-case" procedure is one of the key tenets of the area of robust optimization [12, 20].

Let us first focus on settings when robustification coincides with a regularization problem. In such a case, the robustification identifies the adversarial perturbations the model is protected against, which can in turn provide additional insight into the behavior of different regularizers. Moreover, the adversarial approach is of interest in its own right, even if robustification does not correspond directly to a regularization problem. This is evidenced in part by the burgeoning success of generative adversarial networks and other methodologies in deep learning [116, 117, 242]. Further, the worst-case approach often leads to a more straightforward analysis of properties of estimators [275] as well as algorithms for finding estimators [13].

Let us now return to the robustification problem. A natural choice of an uncertainty set which gives rise to interpretability is the set $\mathcal{U} = \{\Delta \in \mathbb{R}^{n \times p} : \|\Delta\| \leq \lambda\}$, where $\|\cdot\|$ is some matrix norm and $\lambda > 0$. One can then write $\max_{\Delta \in \mathcal{U}} g(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta})$ as

$$\begin{aligned} \max_{\tilde{\mathbf{X}}} \quad & g(\mathbf{y} - \tilde{\mathbf{X}}\boldsymbol{\beta}) \\ \text{s.t.} \quad & \|\mathbf{X} - \tilde{\mathbf{X}}\| \leq \lambda, \end{aligned}$$

or the worst case error taken over all $\tilde{\mathbf{X}}$ sufficiently close to the data matrix \mathbf{X} . In what follows, if $\|\cdot\|$ is a norm or seminorm (see Section 2.1), then we let $\mathcal{U}_{\|\cdot\|}$ denote the ball of radius λ in $\|\cdot\|$:

$$\mathcal{U}_{\|\cdot\|} = \{\Delta : \|\Delta\| \leq \lambda\}.$$

We briefly mention addressing uncertainty in \mathbf{y} . Suppose that we have a set $\mathcal{V} \subseteq \mathbb{R}^n$ which captures some belief about the uncertainty in \mathbf{y} . If again we have an uncertainty set $\mathcal{U} \subseteq \mathbb{R}^{n \times p}$, we may attempt to solve a problem of the form

$$\min_{\boldsymbol{\beta}} \max_{\delta \in \mathcal{V}} g(\mathbf{y} + \delta - (\mathbf{X} + \Delta)\boldsymbol{\beta}).$$

We can instead work with a new loss function \bar{g} defined as

$$\bar{g}(\mathbf{v}) := \max_{\delta \in \mathcal{V}} g(\mathbf{v} + \delta).$$

If g is convex, then so is \bar{g} . In this way, we can work with the problem in the form

$$\min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}} \bar{g}(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}),$$

where there is only uncertainty in \mathbf{X} . Throughout the remainder of the chapter we will only consider such uncertainty.

In the context of linear regression we characterize precisely under which conditions on the model of uncertainty used and on the loss function penalties robustification is equivalent to regularization. In particular we demonstrate that Lasso is equivalent to robustification for an appropriate uncertainty set, which suggests the strong out of sample performance of Lasso can be attributed to its robustness properties, as we provide computational evidence that in general regularization leads to stronger out of sample performance.

2.1 Norms and their Duals

In this section, we introduce the necessary background on norms which we will use to address the equivalence of robustification and regularization in the context of linear regression. Given a vector space $V \subseteq \mathbb{R}^n$ we say that $\|\cdot\| : V \rightarrow \mathbb{R}$ is a *norm* if for all $\mathbf{v}, \mathbf{w} \in V$ and $\alpha \in \mathbb{R}$

1. If $\|\mathbf{v}\| = 0$, then $\mathbf{v} = 0$,
2. $\|\alpha\mathbf{v}\| = |\alpha|\|\mathbf{v}\|$ (absolute homogeneity), and
3. $\|\mathbf{v} + \mathbf{w}\| \leq \|\mathbf{v}\| + \|\mathbf{w}\|$ (triangle inequality).

If $\|\cdot\|$ satisfies Conditions 2 and 3, but not 1, we call it a *seminorm*. For a norm $\|\cdot\|$ on \mathbb{R}^n we define its dual, denoted $\|\cdot\|_*$, to be

$$\|\boldsymbol{\beta}\|_* := \max_{\mathbf{x} \in \mathbb{R}^n} \frac{\mathbf{x}^T \boldsymbol{\beta}}{\|\mathbf{x}\|}.$$

For example, the ℓ_p norms $\|\boldsymbol{\beta}\|_p := (\sum_i |\beta_i|^p)^{1/p}$ for $p \in [1, \infty)$ and $\|\boldsymbol{\beta}\|_\infty := \max_i |\beta_i|$ satisfy a well-known duality relation: ℓ_{p^*} is dual to ℓ_p , where $p^* \in [1, \infty]$ with $1/p + 1/p^* = 1$. We call p^* the *conjugate* of p . More generally for matrix norms¹ $\|\cdot\|$ on $\mathbb{R}^{n \times p}$ the dual is defined analogously:

$$\|\Delta\|_* := \max_{\mathbf{A} \in \mathbb{R}^{n \times p}} \frac{\langle \mathbf{A}, \Delta \rangle}{\|\mathbf{A}\|},$$

where $\Delta \in \mathbb{R}^{n \times p}$ and $\langle \cdot, \cdot \rangle$ denotes the trace inner product: $\langle \mathbf{A}, \Delta \rangle = \text{Tr}(\mathbf{A}^T \Delta)$. We note that the dual of the dual norm is the original norm [51]. Three widely used choices for matrix norms (see [141]) are Frobenius, spectral, and induced norms. The definitions for these norms are given below for $\Delta \in \mathbb{R}^{n \times p}$ and summarized in Table 2.1 for convenient reference.

1. The p -Frobenius norm, denoted $\|\cdot\|_{F_p}$, is the entrywise ℓ_p norm on the entries of Δ :

$$\|\Delta\|_{F_p} := \left(\sum_{ij} |\Delta_{ij}|^p \right)^{1/p}.$$

Analogous to before, F_{p^*} is dual to F_p , where $1/p + 1/p^* = 1$.

2. The p -spectral (Schatten) norm, denoted $\|\cdot\|_{\sigma_p}$, is the ℓ_p norm on the singular values of the matrix Δ :

$$\|\Delta\|_{\sigma_p} := \|\boldsymbol{\mu}(\Delta)\|_p,$$

where $\boldsymbol{\mu}(\Delta)$ denotes the vector containing the singular values of Δ . Again, σ_{p^*} is dual to σ_p .

3. Finally we consider the class of induced norms. If $g : \mathbb{R}^n \rightarrow \mathbb{R}$ and $h : \mathbb{R}^p \rightarrow \mathbb{R}$ are norms, then we define the induced norm $\|\cdot\|_{(h,g)}$ as

$$\|\Delta\|_{(h,g)} := \max_{\boldsymbol{\beta} \in \mathbb{R}^n} \frac{g(\Delta \boldsymbol{\beta})}{h(\boldsymbol{\beta})}.$$

¹We treat a matrix norm as any norm on $\mathbb{R}^{n \times p}$ which satisfies the three conditions of a usual vector norm.

An important special case occurs when $g = \ell_p$ and $h = \ell_q$. When such norms are used, (q, p) is used as shorthand to denote (ℓ_q, ℓ_p) .

Table 2.1: Matrix norms on $\Delta \in \mathbb{R}^{n \times p}$.

Name	Notation	Definition	Description
p-Frobenius	F_p	$\left(\sum_{ij} \Delta_{ij} ^p \right)^{1/p}$	entrywise ℓ_p norm
p-spectral	σ_p	$\ \mu(\Delta)\ _p$	ℓ_p of singular values
Induced	(h, g)	$\max_{\beta} \frac{g(\Delta\beta)}{h(\beta)}$	induced by norms g, h

2.2 Equivalence of Robustification and Regularization

A natural question is when do the procedures of regularization and robustification coincide. In this section, we present settings in which robustification is equivalent to regularization. We begin with a general result on robustification under induced seminorm uncertainty sets.

Theorem 2.1. *If $g : \mathbb{R}^n \rightarrow \mathbb{R}$ is a seminorm which is not identically zero and $h : \mathbb{R}^n \rightarrow \mathbb{R}$ is a norm, then for any $\mathbf{z} \in \mathbb{R}^n$ and $\beta \in \mathbb{R}^p$*

$$\max_{\Delta \in \mathcal{U}_{(h,g)}} g(\mathbf{z} + \Delta\beta) = g(\mathbf{z}) + \lambda h(\beta),$$

where $\mathcal{U}_{(h,g)} = \{\Delta : \|\Delta\|_{(h,g)} \leq \lambda\}$.

Proof. From the triangle inequality $g(\mathbf{z} + \Delta\beta) \leq g(\mathbf{z}) + g(\Delta\beta) \leq g(\mathbf{z}) + \lambda h(\beta)$ for any $\Delta \in \mathcal{U} := \mathcal{U}_{(h,g)}$. We next show that there exists some $\Delta \in \mathcal{U}$ so that $g(\mathbf{z} + \Delta\beta) = g(\mathbf{z}) + \lambda h(\beta)$. Let $\mathbf{v} \in \mathbb{R}^n$ so that $\mathbf{v} \in \operatorname{argmax}_{h^*(\mathbf{v})=1} \mathbf{v}'\beta$, where h^* is the dual norm of h . Note in particular that $\mathbf{v}'\beta = h(\beta)$ by the definition of the dual norm h^* . For now suppose that $g(\mathbf{z}) \neq 0$. Define the rank one matrix $\widehat{\Delta} = \frac{\lambda}{g(\mathbf{z})}\mathbf{z}\mathbf{v}'$. Observe that

$$g(\mathbf{z} + \widehat{\Delta}\beta) = g\left(\mathbf{z} + \frac{\lambda h(\beta)}{g(\mathbf{z})}\mathbf{z}\right) = \frac{g(\mathbf{z}) + \lambda h(\beta)}{g(\mathbf{z})}g(\mathbf{z}) = g(\mathbf{z}) + \lambda h(\beta).$$

We next show that $\widehat{\Delta} \in \mathcal{U}$. Observe that for any $\mathbf{x} \in \mathbb{R}^n$ that

$$g(\widehat{\Delta}\mathbf{x}) = g\left(\frac{\lambda \mathbf{v}' \mathbf{x}}{g(\mathbf{z})}\mathbf{z}\right) = \lambda |\mathbf{v}' \mathbf{x}| \leq \lambda h(\mathbf{x})h^*(\mathbf{v}) = \lambda h(\mathbf{x}),$$

where the final inequality follows by definition of the dual norm. Hence $\widehat{\Delta} \in \mathcal{U}$, as desired.

We now consider the case when $g(\mathbf{z}) = 0$. Let $\mathbf{u} \in \mathbb{R}^n$ so that $g(\mathbf{u}) = 1$ (because g is not identically zero there exists some \mathbf{u} so that $g(\mathbf{u}) > 0$, and so by homogeneity of g we can take \mathbf{u} so that $g(\mathbf{u}) = 1$). Let \mathbf{v} be as before. Now define $\widehat{\Delta} = \lambda \mathbf{u} \mathbf{v}'$. We observe that

$$g(\mathbf{z} + \widehat{\Delta}\beta) = g(\mathbf{z} + \lambda \mathbf{u} \mathbf{v}'\beta) \leq g(\mathbf{z}) + \lambda |\mathbf{v}'\beta| g(\mathbf{u}) = \lambda h(\beta).$$

Now, by the reverse triangle inequality,

$$g(\mathbf{z} + \widehat{\Delta}\beta) \geq g(\widehat{\Delta}\beta) - g(\mathbf{z}) = g(\widehat{\Delta}\beta) = \lambda h(\beta),$$

and therefore $g(\mathbf{z} + \widehat{\Delta}\beta) = \lambda h(\beta) = g(\mathbf{z}) + \lambda h(\beta)$. The proof that $\widehat{\Delta} \in \mathcal{U}$ is identical to the case when $g(\mathbf{z}) \neq 0$. \square

As a corollary we obtain:

Corollary 1. *If $p, q \in [1, \infty]$ then*

$$\min_{\beta} \max_{\Delta \in \mathcal{U}_{(q,p)}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_p = \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_p + \lambda \|\beta\|_q.$$

In particular, for $p = q = 2$ we recover regularized least squares as a robustification; likewise, for $p = 2$ and $q = 1$ we recover the Lasso.²

Theorem 2.2. *One has the following for any $p, q \in [1, \infty]$:*

$$\min_{\beta} \max_{\Delta \in \mathcal{U}_{F_p}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_p = \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_p + \lambda \|\beta\|_{p^*},$$

where p^* is the conjugate of p . Similarly,

$$\min_{\beta} \max_{\Delta \in \mathcal{U}_{\sigma_q}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_2 = \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2 + \lambda \|\beta\|_2.$$

Observe that regularized least squares arises again under all uncertainty sets defined by the spectral norms σ_q when the loss function is $g = \ell_2$. As per Corollary 1 Lasso arises as uncertain ℓ_2 regression with uncertainty set $\mathcal{U} := \mathcal{U}_{(1,2)}$. As with Theorem 2.1, one might argue that the ℓ_1 penalizer arises as an artifact of the model of uncertainty. We remark that one can derive the set \mathcal{U} as an induced uncertainty set defined using the “true” non-convex penalty ℓ_0 , where $\|\beta\|_0 := |\{i : \beta_i \neq 0\}|$. To be precise, for any $p \in [1, \infty]$ and for $\Gamma = \{\beta \in \mathbb{R}^n : \|\beta\|_p \leq 1\}$ we claim that

$$\mathcal{U}' := \left\{ \Delta : \max_{\beta \in \Gamma} \frac{\|\Delta\beta\|_2}{\|\beta\|_0} \leq \lambda \right\}$$

²Strictly speaking, we recover *equivalent* problems to regularized least squares and Lasso, respectively. We take the usual convention and overlook this technicality (see [12] for a discussion). For completeness, we note that one can work directly with the true ℓ_2^2 loss function, although at the cost of requiring more complicated uncertainty sets to recover equivalence results.

satisfies $\mathcal{U} = \mathcal{U}'$. This is summarized, with an additional representation \mathcal{U}'' , in the following proposition.

Proposition 1. *If $\mathcal{U} = \mathcal{U}_{(1,2)}$, $\mathcal{U}' = \{\Delta : \|\Delta\beta\|_2 \leq \lambda \|\beta\|_0 \forall \|\beta\|_p \leq 1\}$ for an arbitrary $p \in [1, \infty]$, and $\mathcal{U}'' = \{\Delta : \|\Delta_i\|_2 \leq \lambda \forall i\}$, where Δ_i is the i th column of Δ , then $\mathcal{U} = \mathcal{U}' = \mathcal{U}''$.*

Proof. We first show that $\mathcal{U} = \mathcal{U}'$. Because $\|\beta\|_1 \leq \|\beta\|_0$ for all $\beta \in \mathbb{R}^n$ with $\|\beta\|_p \leq 1$, we have that $\mathcal{U} \subseteq \mathcal{U}'$. Now suppose that $\Delta \in \mathcal{U}'$. Then for any $\beta \in \mathbb{R}^n$, we have that

$$\|\Delta\beta\|_2 = \left\| \sum_i \beta_i \Delta e_i \right\|_2 \leq \sum_i |\beta_i| \|\Delta e_i\|_2 \leq \sum_i |\beta_i| \lambda = \lambda \|\beta\|_1,$$

where $\{e_i\}_{i=1}^n$ is the standard orthonormal basis for \mathbb{R}^n . Hence, $\Delta \in \mathcal{U}$ and therefore $\mathcal{U}' \subseteq \mathcal{U}$. Combining with the previous direction gives $\mathcal{U} = \mathcal{U}'$.

We now prove that $\mathcal{U} = \mathcal{U}''$. That $\mathcal{U}'' \subseteq \mathcal{U}$ is essentially obvious; $\mathcal{U} \subseteq \mathcal{U}''$ follows by considering $\beta \in \{e_i\}_{i=1}^n$. \square

This proposition implies that ℓ_1 arises from the robustification setting without directly appealing to standard convexity arguments for why ℓ_1 should be used to replace ℓ_0 (which use the fact that ℓ_1 is the so-called convex envelope of ℓ_0 on $[-1, 1]^n$, see e.g. [51]).

In light of the above discussion, it is not difficult to show that other Lasso-like methods can also be expressed as an adversarial robustification, supporting the flexibility and versatility of such an approach. One such example is the elastic net [283, 79, 197], a hybridized version of ridge regression and the Lasso. An equivalent representation of the elastic net is as follows:

$$\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2 + \lambda \|\beta\|_1 + \mu \|\beta\|_2.$$

As per Theorem 2.2, this can be written exactly as

$$\min_{\beta} \max_{\substack{\Delta, \Delta' : \\ \|\Delta\|_{F_\infty} \leq \lambda \\ \|\Delta'\|_{F_2} \leq \mu}} \|\mathbf{y} - (\mathbf{X} + \Delta + \Delta')\beta\|_2.$$

Under this interpretation, we see that λ and μ directly control the tradeoff between two different types of perturbations: “feature-wise” perturbations Δ (controlled via λ and the F_∞ norm) and “global” perturbations Δ' (controlled via μ and the F_2 norm).

We conclude with another example of when robustification is equivalent to regularization for the case of LAD (ℓ_1) and maximum absolute deviation (ℓ_∞) regression under row-wise uncertainty.

Theorem 2.3. *Fix $q \in [1, \infty]$ and let $\mathcal{U} = \{\Delta : \|\delta_i\|_q \leq \lambda \forall i\}$, where δ_i is the i th row of $\Delta \in \mathbb{R}^{n \times p}$. Then*

$$\min_{\beta} \max_{\Delta \in \mathcal{U}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_1 = \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_1 + m\lambda \|\beta\|_{q^*}$$

and

$$\min_{\beta} \max_{\Delta \in \mathcal{U}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_{\infty} = \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_{\infty} + \lambda \|\beta\|_{q^*}.$$

For completeness, we note that the uncertainty set $\mathcal{U} = \{\Delta : \|\delta_i\|_q \leq \lambda \forall i\}$ considered in Theorem 2.3 is actually an induced uncertainty set, namely, $\mathcal{U} = \mathcal{U}_{(q^*, \infty)}$.

2.3 Non-equivalence of robustification and regularization

In this section, we characterize the equivalence between robustification and regularization. We begin with a regularization upper bound on robustification problems.

Proposition 2. *Let $\mathcal{U} \subseteq \mathbb{R}^{n \times p}$ be any non-empty, compact set and $g : \mathbb{R}^n \rightarrow \mathbb{R}$ a seminorm. Then there exists some seminorm $\bar{h} : \mathbb{R}^n \rightarrow \mathbb{R}$ so that for any $\mathbf{z} \in \mathbb{R}^n$, $\beta \in \mathbb{R}^p$,*

$$\max_{\Delta \in \mathcal{U}} g(\mathbf{z} + \Delta\beta) \leq g(\mathbf{z}) + \bar{h}(\beta),$$

with equality when $\mathbf{z} = \mathbf{0}$.

Proof. Let $\bar{h} : \mathbb{R}^n \rightarrow \mathbb{R}$ be defined as

$$\bar{h}(\beta) := \max_{\Delta \in \mathcal{U}} g(\Delta\beta).$$

To show that \bar{h} is a seminorm we must show it satisfies absolute homogeneity and the triangle inequality. For any $\beta \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$,

$$\bar{h}(\alpha\beta) = \max_{\Delta \in \mathcal{U}} g(\Delta(\alpha\beta)) = \max_{\Delta \in \mathcal{U}} |\alpha| g(\Delta\beta) = |\alpha| \left(\max_{\Delta \in \mathcal{U}} g(\Delta\beta) \right) = |\alpha| \bar{h}(\beta),$$

so absolute homogeneity is satisfied. Similarly, if $\gamma \in \mathbb{R}^n$,

$$\begin{aligned} \bar{h}(\beta + \gamma) &= \max_{\Delta \in \mathcal{U}} g(\Delta(\beta + \gamma)) \leq \max_{\Delta \in \mathcal{U}} [g(\Delta\beta) + g(\Delta\gamma)] \\ &\leq \left(\max_{\Delta \in \mathcal{U}} g(\Delta\beta) \right) + \left(\max_{\Delta \in \mathcal{U}} g(\Delta\gamma) \right), \end{aligned}$$

and hence the triangle inequality is satisfied. Therefore, \bar{h} is a seminorm which satisfies the desired properties, completing the proof. \square

When equality is attained for all pairs $(\mathbf{z}, \beta) \in \mathbb{R}^n \times \mathbb{R}^p$, we are in the regime of the previous section, and we say that robustification under \mathcal{U} is equivalent to regularization under \bar{h} . We now discuss a variety of explicit settings in which regularization only provides upper and lower bounds to the true robustified problem.

Fix $p, q \in [1, \infty]$. Consider the robust ℓ_p regression problem

$$\min_{\beta} \max_{\Delta \in \mathcal{U}_{F_q}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_p,$$

where $\mathcal{U}_{F_q} = \{\Delta \in \mathbb{R}^{n \times p} : \|\Delta\|_{F_q} \leq \lambda\}$. In the case when $p = q$ we saw in Theorem 2.2 that one exactly recovers ℓ_p regression with an ℓ_{p^*} penalty:

$$\min_{\beta} \max_{\Delta \in \mathcal{U}_{F_p}} \|\mathbf{y} - (\mathbf{X} + \Delta)\beta\|_p = \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_p + \lambda \|\beta\|_{p^*}.$$

Let us now consider the case when $p \neq q$. We claim that regularization (with \bar{h}) is no longer equivalent to robustification (with \mathcal{U}_{F_q}) unless $p \in \{1, \infty\}$. Applying Proposition 2, one has for any $\mathbf{z} \in \mathbb{R}^n$ that

$$\max_{\Delta \in \mathcal{U}_{F_q}} \|\mathbf{z} + \Delta\beta\|_p \leq \|\mathbf{z}\|_p + \bar{h}(\beta),$$

where $\bar{h} = \max_{\Delta \in \mathcal{U}_{F_q}} \|\Delta\beta\|_p$ is a norm (when $p = q$, this is precisely the ℓ_{p^*} norm, multiplied by λ). Here we can compute \bar{h} . To do this we first define a discrepancy function as follows:

Definition 1. For $a, b \in [1, \infty]$ define the discrepancy function $\delta_m(a, b)$ as

$$\delta_m(a, b) := \max\{\|\mathbf{u}\|_a : \mathbf{u} \in \mathbb{R}^n, \|\mathbf{u}\|_b = 1\}.$$

This discrepancy function is computable and well-known (see e.g. [141]):

$$\delta_m(a, b) = \begin{cases} m^{1/a-1/b}, & \text{if } a \leq b, \\ 1, & \text{if } a > b. \end{cases}$$

It satisfies $1 \leq \delta_m(a, b) \leq m$ and $\delta_m(a, b)$ is continuous in a and b . One has that $\delta_m(a, b) = \delta_m(b, a) = 1$ if and only if $a = b$ (so long as $m \geq 2$). Using this, we now proceed with the theorem. The proof can be found in [25].

Theorem 2.4. The following results hold

(a) For any $\mathbf{z} \in \mathbb{R}^n$ and $\beta \in \mathbb{R}^n$,

$$\max_{\Delta \in \mathcal{U}_{F_q}} \|\mathbf{z} + \Delta\beta\|_p \leq \|\mathbf{z}\|_p + \lambda \delta_m(p, q) \|\beta\|_{q^*}. \quad (2.3)$$

(b) When $p \in \{1, \infty\}$, there is equality in (2.3) for all (\mathbf{z}, β) .

(c) When $p \in (1, \infty)$ and $p \neq q$, for any $\beta \neq \mathbf{0}$ the set of $\mathbf{z} \in \mathbb{R}^n$ for which the inequality (2.3) holds at equality is a finite union of one-dimensional subspaces (so long as $m \geq 2$). Hence, for any $\beta \neq \mathbf{0}$ the inequality in (2.3) is strict for almost all \mathbf{z} .

(d) For $p \in (1, \infty)$, one has for all $\mathbf{z} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$ that

$$\|\mathbf{z}\|_p + \frac{\lambda}{\delta_m(q, p)} \|\boldsymbol{\beta}\|_{q^*} \leq \max_{\Delta \in \mathcal{U}_{F_q}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p. \quad (2.4)$$

(e) For $p \in (1, \infty)$, the lower bound in (2.4) is best possible in the sense that the gap can be arbitrarily small, i.e., for any $\boldsymbol{\beta} \in \mathbb{R}^n$,

$$\inf_{\mathbf{z}} \left(\max_{\Delta \in \mathcal{U}_{F_q}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p - \|\mathbf{z}\|_p - \frac{\lambda}{\delta_m(q, p)} \|\boldsymbol{\beta}\|_{q^*} \right) = 0.$$

Theorem 2.4 characterizes precisely when robustification under \mathcal{U}_{F_q} is equivalent to regularization for the case of ℓ_p regression. In particular, when $p \neq q$ and $p \in (1, \infty)$, the two are *not* equivalent, and one only has that

$$\begin{aligned} \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_p + \frac{\lambda}{\delta_m(q, p)} \|\boldsymbol{\beta}\|_{q^*} &\leq \min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}_{F_q}} \|\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}\|_p \\ &\leq \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_p + \lambda \delta_m(p, q) \|\boldsymbol{\beta}\|_{q^*}. \end{aligned}$$

Further, we have shown that these upper and lower bounds are the *best possible* (Theorem 2.4, parts (c) and (e)). While ℓ_p regression with uncertainty set \mathcal{U}_{F_q} for $p \neq q$ and $p \in (1, \infty)$ still has both upper and lower bounds which correspond to regularization (with different regularization parameters $\bar{\lambda} \in [\lambda/\delta_m(q, p), \lambda\delta_m(p, q)]$), we emphasize that in this case there is no longer the direct connection between the parameter garnering the magnitude of uncertainty (λ) and the parameter for regularization ($\bar{\lambda}$).

Example 2.1. As a concrete example, consider the implications of Theorem 2.4 when $p = 2$ and $q = \infty$. We have that

$$\begin{aligned} \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2 + \lambda \|\boldsymbol{\beta}\|_1 &\leq \min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}_{F_\infty}} \|\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}\|_p \\ &\leq \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2 + \sqrt{m} \lambda \|\boldsymbol{\beta}\|_1. \end{aligned}$$

In this case, robustification is not equivalent to regularization. In particular, in the regime where there are many data points (i.e. m is large), the gap appearing between the different problems can be quite large.

Let us remark that in general, lower bounds on $\max_{\Delta \in \mathcal{U}} g(\mathbf{z} + \Delta \boldsymbol{\beta})$ will depend on the structure of \mathcal{U} and may not exist (except for the trivial lower bound of $g(\mathbf{z})$) in some scenarios. However, it is easy to show that if \mathcal{U} is compact and zero is in the interior of \mathcal{U} , then there exists some $\underline{\lambda} \in (0, 1]$ so that

$$\max_{\Delta \in \mathcal{U}} g(\mathbf{z} + \Delta \boldsymbol{\beta}) \geq g(\mathbf{z}) + \underline{\lambda} \bar{h}(\boldsymbol{\beta}).$$

Before proceeding with other choices of uncertainty sets, it is important to make a further distinction about the general non-equivalence of robustification and regularization as presented in Theorem 2.4. In particular, it is simple to construct examples, which imply the following strong existential result:

Theorem 2.5. *In a setting when robustification and regularization are not equivalent, it is possible for the two problems to have different optimal solutions. In particular,*

$$\boldsymbol{\beta}^* \in \operatorname{argmin}_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}} g(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta})$$

is not necessarily a solution of

$$\min_{\boldsymbol{\beta}} g(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \tilde{\lambda}\bar{h}(\boldsymbol{\beta})$$

for any $\tilde{\lambda} > 0$, and vice versa.

As a result, when robustification and regularization do not coincide, they can induce structurally distinct solutions. In particular, the regularization path (as $\tilde{\lambda} \in (0, \infty)$ varies) and the robustification path (as the radius $\lambda \in (0, \infty)$ of \mathcal{U} varies) can be different.

We now proceed to analyze another setting in which robustification is not equivalent to regularization. The setting, in line with Theorem 2.2, is ℓ_p regression under spectral uncertainty sets \mathcal{U}_{σ_q} . As per Theorem 2.2, one has that

$$\min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}_{\sigma_q}} \|\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}\|_2 = \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2 + \lambda \|\boldsymbol{\beta}\|_2$$

for any $q \in [1, \infty]$. This result on the ‘‘universality’’ of RLS under a variety of uncertainty sets relies on the fact that the ℓ_2 norm underlies spectral decompositions; namely, one can write any matrix \mathbf{X} as $\sum_i \mu_i \mathbf{u}_i \mathbf{v}_i'$, where $\{\mu_i\}_i$ are the singular values of \mathbf{X} , $\{\mathbf{u}_i\}_i$ and $\{\mathbf{v}_i\}_i$ are the left and right singular vectors of \mathbf{X} , respectively, and $\|\mathbf{u}_i\|_2 = \|\mathbf{v}_i\|_2 = 1$ for all i .

A natural question is what happens when the loss function ℓ_2 , a modeling choice, is replaced by ℓ_p , where $p \in [1, \infty]$. We claim that for $p \notin \{1, 2, \infty\}$, robustification under \mathcal{U}_{σ_q} is no longer equivalent to regularization. In light of Theorem 2.4 this is not difficult to prove. We find that the choice of $q \in [1, \infty]$, as before, is inconsequential. We summarize this in the following proposition:

Proposition 3. *For any $\mathbf{z} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$,*

$$\max_{\Delta \in \mathcal{U}_{\sigma_q}} \|\mathbf{z} + \Delta\boldsymbol{\beta}\|_p \leq \|\mathbf{z}\|_p + \lambda \delta_m(p, 2) \|\boldsymbol{\beta}\|_2. \quad (2.5)$$

In particular, if $p \in \{1, 2, \infty\}$, there is equality in (2.5) for all $(\mathbf{z}, \boldsymbol{\beta})$. If $p \notin \{1, 2, \infty\}$, then for any $\boldsymbol{\beta} \neq \mathbf{0}$ the inequality in (2.5) is strict for almost all \mathbf{z} (when $m \geq 2$). Further, for $p \notin \{1, 2, \infty\}$ one has the lower bound

$$\|\mathbf{z}\|_p + \frac{\lambda}{\delta_m(2, p)} \|\boldsymbol{\beta}\|_2 \leq \max_{\Delta \in \mathcal{U}_{\sigma_q}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p,$$

whose gap is arbitrarily small for all $\boldsymbol{\beta}$.

Proof. This result is Theorem 2.4 in disguise. This follows by noting that

$$\max_{\Delta \in \mathcal{U}_{\sigma_q}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p = \max_{\Delta \in \mathcal{U}_{F_2}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p$$

and directly applying the preceding results. \square

We now consider a third setting for ℓ_p regression, this time subject to uncertainty $\mathcal{U}_{(q, r)}$; this is a generalized version of the problems considered in Theorems 2.1 and 2.3. From Theorem 2.1 we know that if $p = r$, then

$$\min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}_{(q, p)}} \|\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}\|_p = \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_p + \lambda \|\boldsymbol{\beta}\|_q.$$

Similarly, as per Theorem 2.3, when $r = \infty$ and $p \in \{1, \infty\}$,

$$\min_{\boldsymbol{\beta}} \max_{\Delta \in \mathcal{U}_{(q, \infty)}} \|\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}\|_p = \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_p + \lambda \delta_m(p, \infty) \|\boldsymbol{\beta}\|_q.$$

Given these results, it is natural to inquire what happens for more general choices of induced uncertainty set $\mathcal{U}_{(q, r)}$. As before with Theorem 2.4, we have a complete characterization of the equivalence of robustification and regularization for ℓ_p regression with uncertainty set $\mathcal{U}_{(q, r)}$:

Proposition 4. *For any $\mathbf{z} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$,*

$$\max_{\Delta \in \mathcal{U}_{(q, r)}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p \leq \|\mathbf{z}\|_p + \lambda \delta_m(p, r) \|\boldsymbol{\beta}\|_q. \quad (2.6)$$

In particular, if $p \in \{1, r, \infty\}$, there is equality in (2.5) for all $(\mathbf{z}, \boldsymbol{\beta})$. If $p \in (1, \infty)$ and $p \neq r$, then for any $\boldsymbol{\beta} \neq \mathbf{0}$ the inequality in (2.6) is strict for almost all \mathbf{z} (when $m \geq 2$). Further, for $p \in (1, \infty)$ with $p \neq r$ one has the lower bound

$$\|\mathbf{z}\|_p + \frac{\lambda}{\delta_m(r, p)} \|\boldsymbol{\beta}\|_q \leq \max_{\Delta \in \mathcal{U}_{(q, r)}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p,$$

whose gap is arbitrarily small for all $\boldsymbol{\beta}$.

Proof. The proof follows the argument given in the proof of Theorem 2.4. Here we simply note that now one uses the fact that

$$\max_{\Delta \in \mathcal{U}_{(q, r)}} \|\mathbf{z} + \Delta \boldsymbol{\beta}\|_p = \max_{\|\mathbf{u}\|_r \leq \lambda \|\boldsymbol{\beta}\|_q} \|\mathbf{z} + \mathbf{u}\|_p.$$

\square

We summarize all of the results on linear regression in Table 2.2.

Table 2.2: Summary of equivalencies for robustification with uncertainty set \mathcal{U} and regularization with penalty \bar{h} , where \bar{h} is as given in Proposition 2. Here by equivalence we mean that for all $\mathbf{z} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$, $\max_{\Delta \in \mathcal{U}} g(\mathbf{z} + \boldsymbol{\beta}) = g(\mathbf{z}) + \bar{h}(\boldsymbol{\beta})$, where g is the loss function, i.e., the upper bound \bar{h} is also a lower bound. Here δ_m is as in Theorem 2.4. Throughout $p, q \in [1, \infty]$ and $m \geq 2$. Here $\boldsymbol{\delta}_i$ denotes the i th row of Δ .

Loss function	\mathcal{U}	$\bar{h}(\boldsymbol{\beta})$	Equivalence \Leftrightarrow
seminorm g	$\mathcal{U}_{(h,g)}$ (h norm)	$\lambda h(\boldsymbol{\beta})$	always
ℓ_p	\mathcal{U}_{σ_q}	$\lambda \delta_m(p, 2) \ \boldsymbol{\beta}\ _2$	$p \in \{1, 2, \infty\}$
ℓ_p	\mathcal{U}_{F_q}	$\lambda \delta_m(p, q) \ \boldsymbol{\beta}\ _{q^*}$	$p \in \{1, q, \infty\}$
ℓ_p	$\mathcal{U}_{(q,r)}$	$\lambda \delta_m(p, r) \ \boldsymbol{\beta}\ _q$	$p \in \{1, r, \infty\}$
ℓ_p	$\ \boldsymbol{\delta}_i\ _q \leq \lambda \forall i$	$\lambda m^{1/p} \ \boldsymbol{\beta}\ _{q^*}$	$p \in \{1, \infty\}$

2.4 The Edge of Robust Regression

In this section, we demonstrate using several real-world data sets that regularized regression leads to stronger out-of-sample predictions. The regularized regressions we report in this section are in fact equivalent to robust regression, which gives a natural explanation of the edge in out of sample performance regularized regression enjoys. We compared the regularized regression

$$\min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_p + \lambda \|\boldsymbol{\beta}\|_q, \quad (2.7)$$

for $(p, q) = (1, 1), (1, 2), (2, 1), (2, 2)$ to the classical regression with $\lambda = 0$ in 8 datasets from the UCI Machine Learning Repository. We solved Problem (2.7) with data in the training set, found the best λ^* in the validation set and report the mean squared prediction error of the estimate corresponding to λ^* on the testing set. Classical regression used $p = 2$ and was trained on training data. We repeated the partition to training, validation, testing sets 30 times and report the average results in Table 2.3. In all cases robust regression had an edge over classical regression, with robust regression with $p = 2, q = 2$ being the best in 6/8 cases. The average improvement in the mean squared error of robust regression over classical regression was 4.3%.

2.5 Concluding Remarks

In this chapter, we have shown that robust optimization provides a natural and useful perspective on the developments of modern statistics. In doing so, we have precisely characterized the equivalence of robustification and regularization and showed that the two are equivalent for a narrow set of

Table 2.3: Average out-of-sample mean squared error for real-world datasets. The best performance is denoted in bold font.

Dataset	n	p	Clas.	1-1	1-2	2-1	2-2
Abalone	4177	9	5.74	5.67	5.65	5.63	5.53
Auto MPG	392	8	18.79	18.72	18.70	18.69	18.58
Comp Hard	209	7	2026.00	2014.32	1978.12	1965.75	1925.13
Concrete	1030	8	132.47	131.46	131.32	131.08	129.31
Forest Fi.	517	13	5526.00	5312.18	5229.14	4994.81	5266.40
Housing	506	13	39.80	39.54	39.49	39.42	39.07
Space Sh.	23	4	0.53	0.52	0.51	0.52	0.52
WPBC	46	32	4723.07	4676.20	4657.98	4630.19	4489.20

parameters. We have further shown that regularized regression, provides stronger out of sample performance compared to classical regression. The explanation of this property is that in the situations we have reported computational results regularization is indeed equivalent with robustification, which protects against data perturbations, which in turn naturally leads to improved out of sample results.

2.6 Notes and Sources

The fact that regularization and robustification coincide was first studied in [111] in the context of uncertain least squares problems and has been extended to more general settings in [275, 68] and most comprehensively in [12]. The exact characterization of the equivalence that we presented in this chapter is from [25].

Relation to robust statistics

There has been extensive work in the robust statistics community on statistical methods which perform well in noisy, real-world environments. As noted in [12], the connection between robust optimization and robust statistics is not clear. Instead of modeling noise via a distributional perspective, as is often the case in robust statistics, in this chapter we choose to model it in a deterministic way using uncertainty sets. For a comprehensive description of the theoretical developments in robust statistics in the last half century, see the texts [144, 231] and the surveys [195, 145]. A central aspect of work in robust statistics is the development and use of a more general set of loss functions. This is in contrast to the robust optimization approach, which generally results in the same nominal loss function with a new penalty. For example, while least squares the ℓ_2 loss is known to perform well under Gaussian noise, it does not perform well under other types of noise, such as contaminated Gaussian noise. Indeed, the Gaussian distribution was defined so that least squares is the

optimal method under Gaussian noise [231]. In contrast, a method like LAD regression (the ℓ_1 loss) generally performs better than least squares with errors in \mathbf{y} , but not necessarily errors in the data matrix \mathbf{X} .

Relation to error-in-variable models

Another class of statistical models which are particularly relevant for the work contained herein are error-in-variable models [69]. One approach to such a problem takes the form

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^n, \Delta \in \mathbb{R}^{m \times n}} g(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}) + P(\Delta),$$

where P is a penalty function which takes into account the complexity of possible perturbations Δ to the data matrix \mathbf{X} . A canonical example of such a method is total least squares [114, 187], which can be written for fixed $\tau > 0$ as

$$\min_{\boldsymbol{\beta}, \Delta} \|\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}\|_2 + \tau \|\Delta\|_F.$$

An equivalent way of writing such problems is, instead of penalized form, as constrained optimization problems. In particular, the constrained version generically takes the form

$$\min_{\boldsymbol{\beta}} \min_{\substack{\Delta \\ P(\Delta) \leq \eta}} g(\mathbf{y} - (\mathbf{X} + \Delta)\boldsymbol{\beta}), \quad (2.8)$$

where $\eta > 0$ is fixed. Under the representation in (2.8), the comparison with the robust optimization approach in (2.2) becomes immediate. While the classical error-in-variables approach takes an optimistic view on uncertainty in the data matrix \mathbf{X} , and finds loadings $\boldsymbol{\beta}$ on the new “corrected” data matrix $\mathbf{X} + \Delta$, the minimax approach of (2.2) considers protections against adversarial perturbations in the data which maximally increase the loss.

Chapter 3

Sparse Regression

The purpose of models is not to fit the data but to sharpen the questions.

– Sam Karlin

Contents

- 3.1. A Primal Approach to Sparse Linear Regression
- 3.2. Computational Insights from the Primal Algorithm
- 3.3. A Dual Approach to Sparse Linear Regression
- 3.4. A cutting plane algorithm
- 3.5. Scalability and Phase Transitions
- 3.6. Concluding Remarks
- 3.7. Notes and Sources

Given input data $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathbb{R}^{n \times p}$ and response data $\mathbf{y} = (y_1, \dots, y_n) \in \mathbb{R}^n$, the problem of linear regression with a regularization term [252] and an explicit sparsity constraint is defined as

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & \frac{1}{2} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{1}{2\gamma} \|\boldsymbol{\beta}\|_2^2 \\ \text{s.t.} \quad & \|\boldsymbol{\beta}\|_0 \leq k, \end{aligned} \tag{3.1}$$

where $\gamma > 0$ is a given weight that controls the importance of the regularization term.

If $n < p$, there are infinitely many solutions to (3.1) without the sparsity constraint $\|\boldsymbol{\beta}\|_0 \leq k$, so the problem is not well defined; that is, the sparsity constraint is not a luxury but a necessity. If $n \geq p$, then the linear regression problem is well defined, but the sparsity constraint makes the solution much more interpretable. The regularization term in the objective in (3.1) is motivated, as we saw in Chapter 2, and helps to reduce the effect of noise in the input data.

In this chapter, we present two approaches to address sparsity in linear regression. A primal approach that models the problem directly and a dual approach that reformulates the sparse regression problem (3.1) as a binary convex optimization problem. The primal approach models the problem directly, extends to modeling many other properties of linear regression, which we explore in Chapter 5 and scales for n, p in the 1000s to provable optimality in minutes and to finding high quality solutions for n, p in the 100,000s in seconds. The dual approach is more powerful and scales for n, p in the 100,000s to provable optimality in seconds. This is surprising as problem (3.1) is *NP-hard*. The ability to solve the problem for very high dimensions allows us to observe new phase transition phenomena. Contrary to traditional complexity theory which suggests that the difficulty of a problem increases with problem size, the sparse regression problem has the property that as the number of samples n increases the problem becomes easier in that the solution recovers 100% of the true signal, and it can be solved to provable optimality extremely fast. In contrast, for a small number of samples n , the problem takes a larger amount of time to solve, but importantly the optimal solution provides a statistically more relevant solution than alternative heuristic solutions. We argue that the dual sparse regression approach presents a superior alternative over heuristic methods available at present.

3.1 A Primal Approach to Sparse Linear Regression

In this section, we reformulate (3.1) as a mixed integer quadratic optimization problem, and solve it with commercial solvers exploiting the very significant progress these solvers have made.

We introduce decision variables s_j , $j \in [p]$, which are equal to 1, if variable x_j is selected, and 0, otherwise. Using a big M formulation for the cardinality constraint, the sparse regression problem (3.1) can indeed be transformed into the problem

$$\begin{aligned} \min \quad & \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{1}{2\gamma} \|\boldsymbol{\beta}\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{\beta} \in \mathbb{R}^p, \mathbf{s} \in \mathbb{S}_k^p \\ & -Ms_j \leq \beta_j \leq Ms_j, \quad j = 1, \dots, p, \end{aligned} \quad (3.2)$$

where $\mathbb{S}_k^p = \{\mathbf{s} \in \{0, 1\}^p, \mathbf{e}^T \mathbf{s} \leq k\}$. The constraint in (3.2) ensures that the regression coefficient β_j is nonzero only if the selection variable $s_j = 1$ for a sufficiently large constant M . The choice of this data-dependent constant M affects the strength of the formulation (3.2) and is critical for obtaining solutions quickly in practice. For the case $n \geq p$, upper and lower bounds on the optimal values $\hat{\beta}_j$ to problem (3.1) can be obtained by solving the following pair of convex optimization problems:

$$\begin{aligned} u_j^+ := \max_{\boldsymbol{\beta}} \beta_j \quad (u_j^- := \min_{\boldsymbol{\beta}} \beta_j) \\ \text{s.t.} \quad \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{1}{2\gamma} \|\boldsymbol{\beta}\|_2^2 \leq \text{UB}, \end{aligned} \quad (3.3)$$

where UB is an upper bound to the minimum of problem (3.1). u_j^+ is an upper bound to $\hat{\beta}_j$, since the cardinality constraint $\|\boldsymbol{\beta}\|_0 \leq k$ does not appear in the optimization problem. Similarly, u_j^- is a lower bound to $\hat{\beta}_j$. Then, $M = \max\{|u_j^+|, |u_j^-|\}$ serves as an upper bound to $|\hat{\beta}_j|$. A reasonable choice for UB is obtained by using a heuristic solution for the cardinality-constrained problem.

A Discrete First Order Algorithm

We develop a discrete extension of first order methods in convex optimization to obtain near optimal solutions for Problem (3.2). Let

$$g(\boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{1}{2\gamma} \|\boldsymbol{\beta}\|_2^2,$$

with

$$\nabla g(\boldsymbol{\beta}) = -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \frac{1}{\gamma} \boldsymbol{\beta}.$$

Clearly $g(\boldsymbol{\beta}) \geq 0$, is convex and has Lipschitz continuous gradient:

$$\|\nabla g(\boldsymbol{\beta}) - \nabla g(\tilde{\boldsymbol{\beta}})\| \leq \ell \|\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}\|. \quad (3.4)$$

with $\ell = \lambda_{\max}(\mathbf{X}'\mathbf{X} + \frac{1}{\gamma} \mathbf{I})$. The discrete first order algorithm applies to

$$\min_{\boldsymbol{\beta}} g(\boldsymbol{\beta}) \quad \text{subject to} \quad \|\boldsymbol{\beta}\|_0 \leq k, \quad (3.5)$$

where, $g(\boldsymbol{\beta}) \geq 0$, is convex and satisfies (3.4). We first observe that when $g(\boldsymbol{\beta}) = \|\boldsymbol{\beta} - \mathbf{c}\|_2^2$ for a given \mathbf{c} , Problem (3.5) admits a closed form solution.

Proposition 3.1. *The optimal solution $\hat{\boldsymbol{\beta}}$, denoted by $\mathbf{H}_k(\mathbf{c})$, to*

$$\hat{\boldsymbol{\beta}} \in \arg \min_{\|\boldsymbol{\beta}\|_0 \leq k} \|\boldsymbol{\beta} - \mathbf{c}\|_2^2, \quad (3.6)$$

retains the k largest (in absolute value) elements of $\mathbf{c} \in \mathbb{R}^p$ and sets the rest to zero, i.e., if $|c_{(1)}| \geq |c_{(2)}| \geq \dots \geq |c_{(p)}|$, denote the ordered values of the absolute values of the vector \mathbf{c} , then:

$$\hat{\beta}_i = \begin{cases} c_i, & \text{if } i \in \{(1), \dots, (k)\}, \\ 0, & \text{otherwise.} \end{cases} \quad (3.7)$$

The notation “argmin” (appearing in Problem (3.6) and other places that follow) denotes the set of minimizers. Operator (3.7) is also known as the hard-thresholding operator [84].

Given a current solution $\boldsymbol{\beta}$, the second ingredient of the overall approach is to upper bound the function $g(\boldsymbol{\eta})$ around $g(\boldsymbol{\beta})$. To do so, we use ideas from projected gradient descent methods in first order convex optimization problems.

Proposition 3.2. ([205, 204]) *For a convex function $g(\boldsymbol{\beta})$ satisfying condition (3.4) and for any $L \geq \ell$ we have:*

$$g(\boldsymbol{\eta}) \leq Q_L(\boldsymbol{\eta}, \boldsymbol{\beta}) := g(\boldsymbol{\beta}) + \frac{L}{2} \|\boldsymbol{\eta} - \boldsymbol{\beta}\|_2^2 + \langle \nabla g(\boldsymbol{\beta}), \boldsymbol{\eta} - \boldsymbol{\beta} \rangle \quad (3.8)$$

for all $\boldsymbol{\beta}, \boldsymbol{\eta}$ with equality holding at $\boldsymbol{\beta} = \boldsymbol{\eta}$.

Applying Proposition 3.1 to the upper bound $Q_L(\boldsymbol{\eta}, \boldsymbol{\beta})$ in Proposition 3.2 we obtain

$$\begin{aligned} \arg \min_{\|\boldsymbol{\eta}\|_0 \leq k} Q_L(\boldsymbol{\eta}, \boldsymbol{\beta}) &= \arg \min_{\|\boldsymbol{\eta}\|_0 \leq k} \left(\frac{L}{2} \left\| \boldsymbol{\eta} - \left(\boldsymbol{\beta} - \frac{1}{L} \nabla g(\boldsymbol{\beta}) \right) \right\|_2^2 \right. \\ &\quad \left. - \frac{1}{2L} \|\nabla g(\boldsymbol{\beta})\|_2^2 + g(\boldsymbol{\beta}) \right) \\ &= \arg \min_{\|\boldsymbol{\eta}\|_0 \leq k} \left\| \boldsymbol{\eta} - \left(\boldsymbol{\beta} - \frac{1}{L} \nabla g(\boldsymbol{\beta}) \right) \right\|_2^2 \\ &= \mathbf{H}_k \left(\boldsymbol{\beta} - \frac{1}{L} \nabla g(\boldsymbol{\beta}) \right), \end{aligned} \quad (3.9)$$

where $\mathbf{H}_k(\cdot)$ is defined in (3.7). In light of (3.9) we are now ready to present Algorithm 3.1 to find a first order stationary point $\boldsymbol{\eta} \in \mathbb{R}^p$ of problem (3.5), that is $\boldsymbol{\eta}$ satisfies $\|\boldsymbol{\eta}\|_0 \leq k$ and

$$\boldsymbol{\eta} \in \mathbf{H}_k \left(\boldsymbol{\eta} - \frac{1}{L} \nabla g(\boldsymbol{\eta}) \right). \quad (3.10)$$

Algorithm 3.1 DISCRETE FIRST ORDER ALGORITHM

Input: $g(\beta)$, L , ϵ ; starting solution $\beta_1 \in \mathbb{R}^p$ such that $\|\beta_1\|_0 \leq k$.

Output: A first order stationary solution β^* .

- 1: $m \leftarrow 1$
 - 2: **repeat**
 - 3: $m \leftarrow m + 1$
 - 4: $\beta_m \leftarrow \mathbf{H}_k (\beta_{m-1} - \frac{1}{L} \nabla g(\beta_{m-1}))$
 - 5: **until** $g(\beta_m) - g(\beta_{m-1}) \leq \epsilon$
 - 6: **return** β_m
-

Convergence Analysis of Algorithm 3.1

We next address the asymptotic convergence properties of Algorithm 3.1.

Theorem 3.1 ([38]). *Let $L > \ell$ and β^* denote a first order stationary point of Algorithm 1. After N iterations Algorithm 3.1, satisfies*

$$\min_{m=1,\dots,N} \|\beta_{m+1} - \beta_m\|_2^2 \leq \frac{2(g(\beta_1) - g(\beta^*))}{N(L - \ell)}, \quad (3.11)$$

where $g(\beta_m) \downarrow g(\beta^*)$ as $m \rightarrow \infty$.

Theorem 3.1 implies that for any $\epsilon > 0$ there exists $N = O(\frac{1}{\epsilon})$ such that for some $1 \leq m^* \leq N$, we have: $\|\beta_{m^*+1} - \beta_{m^*}\|_2^2 \leq \epsilon$. Furthermore, the support of β_m stabilizes after finitely many iterations, after which Algorithm 3.1 behaves like gradient descent on the stabilized support. If $g(\beta)$ restricted to this support is strongly convex, which is the case for problem (3.1), then Algorithm 3.1 will enjoy a linear rate of convergence [205], as soon as the support stabilizes.

Polishing coefficients on the active set

Algorithm 3.1 detects the active set after a few iterations. Once the active set stabilizes, the algorithm may take a number of iterations to estimate the values of the regression coefficients on the active set to a high accuracy level. In this context, we found the following simple method to be quite useful. When the algorithm has converged to a tolerance of ϵ ($\approx 10^{-4}$), we fix the current active set, \mathcal{I} , and solve a lower-dimensional convex optimization problem:

$$\min_{\beta} \quad g(\beta) \quad s.t. \quad \beta_i = 0, \quad i \notin \mathcal{I}. \quad (3.12)$$

For the least squares and least absolute deviation problems, Problem (3.12) can be solved very efficiently.

We observed in the computational experiments that Algorithm 3.2, a minor variant of Algorithm 3.1 had better empirical performance.

Algorithm 3.2 DISCRETE FIRST ORDER ALGORITHM WITH LINE SEARCH

Input: $g(\beta)$, L , ϵ ; starting solution $\beta_1 \in \mathbb{R}^p$ such that $\|\beta_1\|_0 \leq k$.

Output: A first order stationary solution β^* .

```

1:  $m \leftarrow 1$ 
2: repeat
3:    $m \leftarrow m + 1$ 
4:    $\eta_m \leftarrow \mathbf{H}_k(\beta_{m-1} - \frac{1}{L}\nabla g(\beta_{m-1}))$ 
5:    $\lambda_m \leftarrow \arg \min_{\lambda} g(\lambda\eta_m + (1-\lambda)\beta_{m-1})$ 
6:    $\beta_m \leftarrow \lambda_m\eta_m + (1-\lambda_m)\beta_{m-1}$ 
7: until  $g(\beta_m) - g(\beta_{m-1}) \leq \epsilon$ 
8: return  $\beta_m$ 
```

Algorithm 3.2 modifies Step 4 of Algorithm 3.1 by using a simple line search.

The overall primal algorithm consists of using Algorithm 3.2 to find a feasible solution that serves as a warm start, and then solving formulation (3.2) by commercial solvers.

3.2 Computational Insights from the Primal Algorithm

In this section, we present a variety of computational experiments to assess the algorithmic and statistical performances of the primal algorithm. We consider both the classical overdetermined case with $n \geq p$ and the high dimensional $p > n$ case. We report results on both synthetic and real-world data.

Synthetic Datasets

We consider a collection of problems where $\mathbf{x}_i \sim N(\mathbf{0}, \Sigma)$, $i = 1, \dots, n$ are independent realizations from a p -dimensional multivariate normal distribution with mean zero and covariance matrix $\Sigma := (\sigma_{ij})$. The columns of the \mathbf{X} matrix were subsequently standardized to have unit ℓ_2 norm. For a fixed \mathbf{X} , we generated \mathbf{y} via $\mathbf{y} = \mathbf{X}\beta^0 + \epsilon$, with $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$. We denote the number of nonzeros in β^0 by k_0 . We define the Signal-to-Noise Ratio (SNR) of the problem as: $\text{SNR} = \text{Var}(\mathbf{x}'\beta^0)/\sigma^2$. We consider the following examples:

Example 3.1. $\sigma_{ij} = \rho^{|i-j|}$ for $i, j \in \{1, \dots, p\} \times \{1, \dots, p\}$. We consider different values of $k_0 \in \{5, 10\}$ and $\beta_i^0 = 1$ for k_0 equi-spaced values; rounding the indices to the nearest large integer value of $i \in \{1, 2, \dots, p\}$ when required.

Example 3.2. $\Sigma = \mathbf{I}_{p \times p}$, $k_0 = 5$ and $\beta^0 = (\mathbf{1}'_{5 \times 1}, \mathbf{0}'_{p-5 \times 1})' \in \mathbb{R}^p$.

Example 3.3. $\Sigma = \mathbf{I}_{p \times p}$, $k_0 = 10$ and $\beta_i^0 = \frac{1}{2} + (10 - \frac{1}{2}) \frac{(i-1)}{k_0}$, $i = 1, \dots, 10$ and $\beta_i^0 = 0, \forall i > 10$ — i.e., a vector with ten nonzero entries, with the nonzero values being equally spaced in the interval $[\frac{1}{2}, 10]$.

Example 3.4. $\Sigma = \mathbf{I}_{p \times p}$, $k_0 = 6$ and $\beta^0 = (-10, -6, -2, 2, 6, 10, \mathbf{0}_{p-6})$, i.e., a vector with six nonzero entries, equally spaced in the interval $[-10, 10]$.

Real-world Datasets

We considered the Diabetes dataset [91] with all the second order interactions included in the model, which resulted in 64 predictors. We reduced the sample size to $n = 350$ by taking a random sample and standardized the response and the columns of the model matrix to have zero means and unit ℓ_2 -norm.

In addition to the above, we also considered a real microarray dataset: the Leukemia data [83], with $n = 72$ binary responses and more than 3000 predictors. We standardized the response and features to have zero means and unit ℓ_2 -norm. We reduced the set of features to 1000 by retaining the features maximally correlated (in absolute value) to the response. From the resulting matrix \mathbf{X} had $n = 72$, $p = 1000$, we generated a semi-synthetic dataset with continuous response as $\mathbf{y} = \mathbf{X}\beta^0 + \epsilon$, where the first five coefficients of β^0 were taken as one and the rest as zero. The noise was distributed as $\epsilon_i \stackrel{\text{iid}}{\sim} N(0, \sigma^2)$, with σ^2 chosen to get a $SNR = 7$. Computations were carried out in a linux 64 bit server—Intel(R) Xeon(R) eight-core processor 1.80GHz, 16 GB of RAM. We used GUROBI [124] for the MIO solver. We used $\gamma = \infty$.

The Overdetermined Regime: $n \geq p$

We considered the following three algorithms:

- (a) Algorithm 3.2 with fifty random initializations. We took the solution corresponding to the best objective value.
- (b) MIO with cold start, i.e., formulation (3.2) with a time limit of 500 seconds.
- (c) MIO with warm start. This was the MIO formulation (3.2) initialized with a solution obtained from (a). The combined run was for a total of 500 seconds.

To compare the different algorithms in terms of the quality of upper bounds, we run for every instance all the algorithms and obtain the best solution value f_* . among them, If f_{alg} denotes the value of the best objective

function for method $\text{alg} \in \{\text{(a), (b), (c)}\}$ we define the relative accuracy of the solution obtained by “ alg ” as:

$$\text{Relative Accuracy} = (f_{\text{alg}} - f_*)/f_* \quad (3.13)$$

Table 3.1 shows results for the Diabetes dataset for different values of k . For each algorithm we report the time taken by it to reach the best objective value during the time of 500 seconds. Using the discrete first order methods in combination with the MIO algorithm resulted in finding the best possible relative accuracy in a matter of a few minutes.

Table 3.1: Quality of upper bounds for Problem (3.2) for the Diabetes dataset, for different values of k . We observe that MIO equipped with warm starts deliver the best upper bounds in the shortest overall times. The run time for the MIO with warm start includes the time taken by the discrete first order method.

k	Discrete First Order		MIO Cold Start		MIO Warm Start	
	Accuracy	Time	Accuracy	Time	Accuracy	Time
9	0.1306	1	0.0036	500	0	346
20	0.1541	1	0.0042	500	0	77
49	0.1915	1	0.0015	500	0	87
57	0.1933	1	0	500	0	2

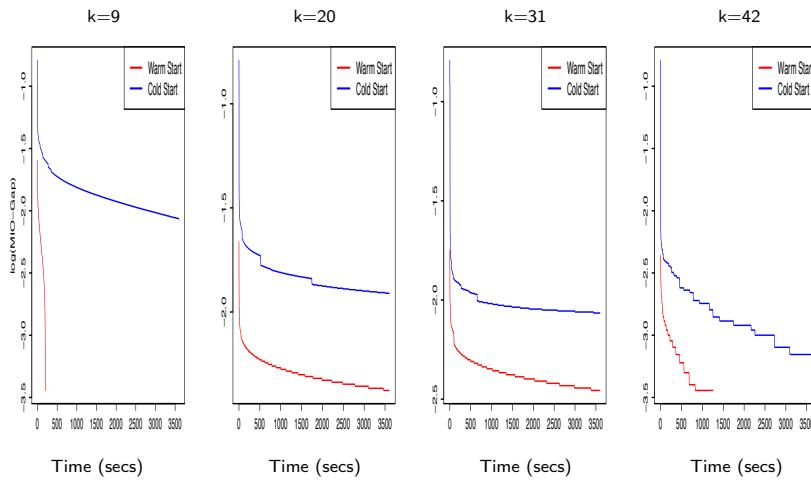
Improving MIO Performance via Warm Starts

We performed a series of experiments on the Diabetes dataset to obtain a globally optimal solution to Problem (3.2) and to understand the implications of using advanced warm starts to the MIO formulation in terms of certifying optimality. For each k , we ran Algorithm 3.2 with fifty random initializations, which took less than a few seconds to run. We used the best solution as an advanced warm start to the MIO formulation (3.2). The MIO solver was provided with problem-specific bounds obtained in (3.3). For each of these examples, we also ran the MIO formulation without any such additional problem-specific information, i.e., formulation (3.3).—we refer to this as “Cold Start”. Figure 3.1 presents a representative subset of the results. We observed that the MIO formulation (3.2) armed with warm-starts and additional bounds closed the optimality gap faster than their “Cold Start” counterpart.

Statistical Performance

We considered datasets as described in Example 3.1, we took different values of n, p with $n \geq p$, ρ with $k_0 = 10$.

Figure 3.1: The evolution of the MIO optimality gap (in $\log_{10}(\cdot)$ scale) for Problem (3.2) for the Diabetes dataset with $n = 350, p = 64$, for different values of k . Here, “Warm Start” indicates that the MIO was provided with warm starts. MIO (“Warm Start”) is found close the optimality gap much faster. In all of these examples, the global optimum was found within a very small fraction of the total time, but the proof of global optimality came later.



Competing Methods and Performance Measures

For every example, we considered the following learning procedures for comparison purposes: (a) the MIO formulation (3.2) equipped with warm starts from Algorithm 3.2 (annotated as “MIO” in the figure), (b) the Lasso, (c) Sparsenet and (d) stepwise regression (annotated as “Step” in the figure). In addition to the above, we have also performed comparisons with an unshrunk version of Lasso under which we performed unrestricted least squares on the Lasso support to mitigate the bias imparted by Lasso.

We used R to compute Lasso, Sparsenet and stepwise regression using the `glmnet`, `Sparsenet` and `Stats` packages respectively.

For each procedure, we obtained the “optimal” tuning parameter by selecting the model that achieved the best predictive performance on a held out validation set. Once the model $\hat{\beta}$ was selected, we obtained the prediction error as:

$$\text{Prediction Error} = \|\mathbf{X}\hat{\beta} - \mathbf{X}\beta^0\|_2^2 / \|\mathbf{X}\beta^0\|_2^2. \quad (3.14)$$

Note that, if the (sample) features are highly correlated, the selected model, may decide to choose a feature instead of its correlated surrogate—in such cases, variable selection error, measured in terms of Hamming distance with respect to the data-generating model, may be misleading. Size of the

optimal model selected serves as a measure of the number of redundant variables selected by the model; and prediction error measures good data-fidelity. We report “prediction error” and number of nonzeros in the optimal model results. The results were averaged over ten random instances: for every run, the training and validation data had a fixed \mathbf{X} but random noise ϵ .

Figure 3.2 presents results for data generated as per Example 3.1 with $n = 500$ and $p = 100$. We see that the MIO procedure performs very well across all the examples. Among the methods, MIO performs the best, followed by **Sparsenet**, **Lasso** with **Step(wise)** exhibiting the worst performance — MIO consistently chose the sparsest model. **Lasso** delivers quite dense models and pays the price in predictive performance too, by selecting wrong variables. As the value of SNR increases, the predictive power of the methods improve, as expected. The differences in predictive errors between the methods diminish with increasing SNR values. With increasing values of ρ (from left panel to right panel in the figure), the number of nonzeros selected by the **Lasso** in the optimal model increases.

We also performed experiments with the unshrunk version of **Lasso**. The unrestricted least squares solution on the optimal model selected by **Lasso** (as shown in Figure 3.2) had worse predictive performance than **Lasso**, with the same sparsity pattern. This is probably due to overfitting since the model selected by **Lasso** is quite dense compared to n, p . We also tried variants of unshrunk **Lasso** which led to models with better performances than **Lasso** but the results were inferior compared to MIO.

The High-Dimensional Regime: $p > n$

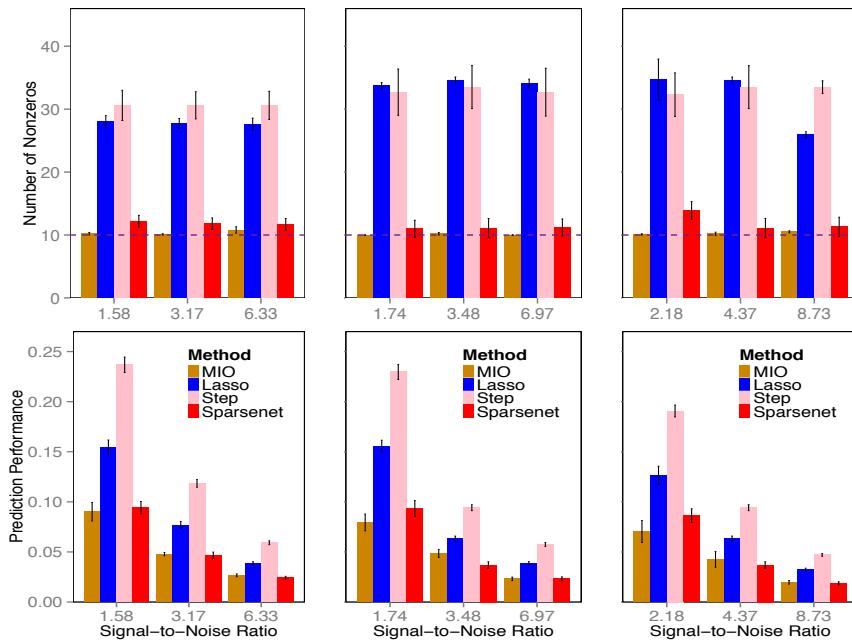
We investigate **(a)** the evolution of upper bounds in the high-dimensional regime; **(b)** the statistical performance of the MIO approach in comparison to other sparse learning methods.

Obtaining Good Upper Bounds

We performed experiments similar to those in the overdetermined Rregime, demonstrating the effectiveness of warm-starting MIO solvers with discrete first order methods. We considered a synthetic dataset corresponding to Example 3.2 with $n = 30, p = 2000$ for varying SNR values (see Table 3.2) over a time of 500s. As before, using the discrete first order methods in combination with the MIO formulation resulted in finding the best possible upper bounds in the shortest possible times.

Figure 3.3 shows the evolution of the objective value of Problem (3.2) for different values of k and $\gamma = \infty$ for the Leukemia dataset. For each k , we warm-started the MIO solver for formulation (3.2) with the solution obtained by Algorithm 3.2 and allowed the MIO solver to run for 4000 seconds—the resultant solution is denoted by f_* . We plot Relative

Figure 3.2: Figure showing the sparsity (upper panel) and predictive performances (bottom panel) for different subset selection procedures for the least squares loss. Here, we consider data generated as per Example 3.1, with $n = 500, p = 100, k_0 = 10$, for three different SNR values with [Left Panel] $\rho = 0.5$, [Middle Panel] $\rho = 0.8$, and [Right Panel] $\rho = 0.9$. The dashed line in the top panel represents the true number of nonzero values. For each of the procedures, the optimal model was selected as the one which produced the best prediction accuracy on a separate validation set.



Accuracy, i.e., $(f_t - f_*)/f_*$, where f_t is the objective value obtained after t seconds. The figure shows that the solution obtained by Algorithm 3.2 is improved by the MIO on various instances and the time taken to improve the upper bounds depends upon k . In general, for smaller values of k the upper bounds obtained by the MIO algorithm stabilize earlier, i.e., MIO finds improved solutions faster than larger values of k .

Statistical Performance

To understand the statistical behavior of MIO when compared to other approaches for learning sparse models, we considered synthetic datasets for values of n ranging from 30 – 50 and values of p ranging from 1000 – 2000. The following methods were used for comparison purposes (a)

Table 3.2: The quality of upper bounds for Problem (3.2) obtained by Algorithm 3.2, MIO with cold start and MIO warm-started with Algorithm 3.2. We consider Example 3.2 with $n = 30, p = 2000$ and different values of SNR. The MIO method, when warm-started with the first order solution performs the best in terms of getting a good upper bound in the shortest time. Here, “Accuracy” is the same metric as defined in (3.13). The first order methods work well, but need not lead to best quality solutions on their own. MIO improves the quality of upper bounds delivered by the first order methods and their combined effect leads to the best performance.

	k	Discrete First Order Accuracy	Time	MIO Cold Start Accuracy	Time	MIO Warm Start Accuracy	Time
SNR = 3	5	0.1647	37.2	1.0510	500	0	72.2
	6	0.6152	41.1	0.2769	500	0	77.1
	7	0.7843	40.7	0.8715	500	0	160.7
	8	0.5515	38.8	2.1797	500	0	295.8
	9	0.7131	45.0	0.4204	500	0	96.0
SNR = 7	5	0.5072	45.6	0.7737	500	0	65.6
	6	1.3221	40.3	0.5121	500	0	82.3
	7	0.9745	40.9	0.7578	500	0	210.9
	8	0.8293	40.5	1.8972	500	0	262.5
	9	1.1879	44.2	0.4515	500	0	254.2

Algorithm 3.2. Here we used fifty different random initializations around $\mathbf{0}$, of the form $\min(i-1, 1)N(\mathbf{0}_{p \times 1}, 4\mathbf{I}), i = 1, \dots, 50$ and took the solution corresponding to the best objective value; **(b)** The MIO approach with warm starts from part (a); **(c)** The Lasso solution and **(d)** The Sparsenet solution.

For methods (a), (b) we considered ten equi-spaced values of k in the range $[3, 2k_0]$ (including the optimal value of k_0). For each of the methods, the best model was selected. For some of the above examples, we also study the performance of the unshrunk version of Lasso. In Figure 3.4 and Figure 3.5 we present selected representative results.

In Figure 3.4 the left panel shows the performance of different methods for Example 3.1 with $n = 50, p = 1000, \rho = 0.8, k_0 = 5$. In this example, there are five nonzero coefficients: the features corresponding to the nonzero coefficients are weakly correlated and a feature having a nonzero coefficient is highly correlated with a feature having a zero coefficient. In this situation, the Lasso selects a very dense model since it fails to distinguish between a zero and a nonzero coefficient when the variables are correlated—it brings both the coefficients in the model (with shrinkage). MIO (with warm-start) performs the best—both in terms of predictive accuracy and in selecting a sparse set of coefficients. MIO

Figure 3.3: Behavior of MIO aided with warm start in obtaining good upper bounds for the Leukemia dataset ($n = 72, p = 1000$). The vertical axis shows relative accuracy, i.e., $(f_t - f_*)/f_*$, where f_t is the objective value after t seconds and f_* denotes the best objective value obtained by the method after 4000 seconds. The colored diamonds correspond to the locations where the MIO (with warm start) attains the best solution. Note that MIO improves the solution obtained by the first order method in all the instances.

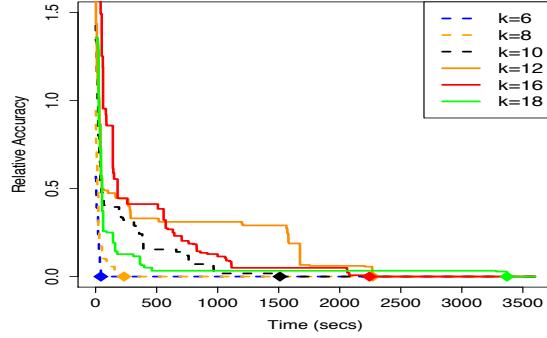
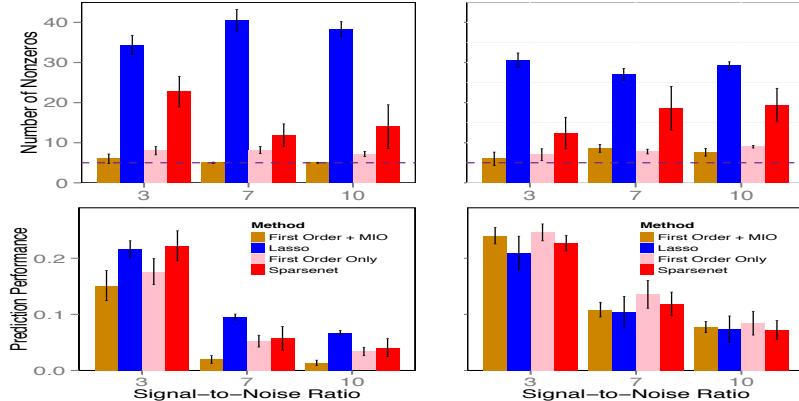
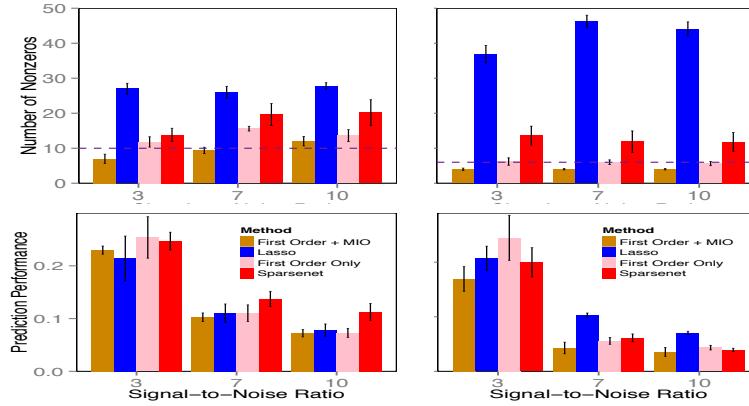


Figure 3.4: The sparsity and predictive performance for different procedures: [Left Panel] shows Example 3.1 with $n = 50, p = 1000, \rho = 0.8, k_0 = 5$ and [Right Panel] shows Example 3.2 with $n = 30, p = 1000$ —for each instance several SNR values have been shown.



obtains the sparsest model among the four methods and seems to find better solutions in terms of statistical properties than the models obtained by the first order methods alone. Interestingly, the “optimal model” selected by the first order methods is more dense than that selected by the MIO. The

Figure 3.5: [Left Panel] Shows performance for data generated according to Example 3.3 with $n = 30, p = 1000$ and [Right Panel] shows Example 3.4 with $n = 50, p = 2000$.



number of nonzero coefficients selected by MIO remains fairly stable across different SNR values, unlike the other three methods. For this example, we also experimented with the different versions of unshrunk **Lasso** and observed that the best unshrunk **Lasso** models had performance marginally better than **Lasso** but quite inferior to MIO.

Figure 3.4 [right panel] shows Example 3.2, with $n = 30, p = 1000, k_0 = 5$ and all nonzero coefficients equal one. Here, all methods perform similarly in terms of predictive accuracy. In fact, for the smallest value of SNR, **Lasso** achieves the best predictive model. In all the cases however, the MIO achieves the sparsest model with favorable predictive accuracy.

In Figure 3.5, for both examples, the model matrix is an iid Gaussian ensemble. The underlying regression coefficient β^0 however, is structurally different than Example 3.2 (as in Figure 3.4, right-panel). The alternating signs of β^0 is responsible for different statistical behaviors of the four methods across Figures 3.4 (right-panel) and Figure 3.5 (both panels). The MIO (with warm-starts) seems to be the best among all the methods. For Example 3.3 (Figure 3.5, left panel) the predictive performances of **Lasso** and MIO are comparable—the MIO however delivers much sparser models than the **Lasso**.

The key conclusions of the computational experiments are as follows:

1. The primal MIO algorithm has a significant edge in detecting the correct sparsity structure for all examples compared to **Lasso**, **Sparsenet** and the stand-alone discrete first order method.
2. For data generated as per Example 3.1 with large values of ρ , the MIO algorithm gives better predictive performance compared to its

competitors.

3. For data generated as per Examples 3.2 and 3.3, MIO delivers similar predictive models like the **Lasso**, but produces much sparser models. In fact, **Lasso** seems to perform marginally better than MIO, as a predictive model for small values of SNR.
4. For Example 3.4, MIO performs the best both in terms of predictive accuracy and delivering sparse models.

3.3 A Dual Approach to Sparse Linear Regression

In this section, we take a different approach to the sparse regression problem (3.1) entirely. To that end we first briefly return to the ordinary regression problem for which any sparsity considerations are ignored and in which a linear relationship between input data \mathbf{X} and observations \mathbf{y} is determined through solving the least squares regression problem

$$\begin{aligned} c = \min & \quad \frac{1}{2\gamma} \|\boldsymbol{\beta}\|_2^2 + \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 \\ \text{s.t. } & \quad \boldsymbol{\beta} \in \mathbb{R}^p. \end{aligned} \tag{3.15}$$

We will refer to the previously defined quantity c as the regression loss. The quantity c does indeed agree with the regularized empirical regression loss for the optimal linear regressor corresponding to the input data \mathbf{X} and response \mathbf{y} . We point out now that the regression loss function c is convex as a function of the outer product \mathbf{XX}^T and furthermore show that it admits an explicit characterization as a semidefinite representable function.

Lemma 3.1 (The regression loss function c). *The regression loss c admits the following explicit characterizations*

$$c = \frac{1}{2} \mathbf{y}^T \left(\mathbf{I}_n - \mathbf{X} \left(\mathbf{I}_p/\gamma + \mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \right) \mathbf{y}, \tag{3.16}$$

$$= \frac{1}{2} \mathbf{y}^T \left(\mathbf{I}_n + \gamma \mathbf{XX}^T \right)^{-1} \mathbf{y}. \tag{3.17}$$

Futhermore, the regression loss c as a function of the kernel matrix \mathbf{XX}^T is conic representable using the formulation

$$c(\mathbf{XX}^T) = \min \eta \text{ s.t. } \eta \geq 0, \quad \begin{pmatrix} 2\eta & \mathbf{y}^T \\ \mathbf{y} & \mathbf{I}_n + \gamma \mathbf{XX}^T \end{pmatrix} \in \mathbf{S}_+^{n+1}. \tag{3.18}$$

Proof. As the minimization problem (3.15) over $\boldsymbol{\beta}$ in \mathbb{R}^p is an unconstrained quadratic optimization problem (QOP), the optimal value $\boldsymbol{\beta}^*$ satisfies the linear relationship $(\mathbf{I}_p/\gamma + \mathbf{X}^T \mathbf{X})\boldsymbol{\beta}^* = \mathbf{X}^T \mathbf{y}$. Substituting the expression for

the optimal linear regressor β^* back into optimization problem, we arrive at

$$c = \frac{1}{2}\mathbf{y}^T\mathbf{y} - \frac{1}{2}\mathbf{y}^T\mathbf{X}(\mathbf{I}_p/\gamma + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

establishing the first explicit characterization (3.16) of the regression function c . The second characterization (3.17) can be derived from the first with the help of the matrix inversion lemma

$$(\mathbf{I}_n + \gamma\mathbf{X}\mathbf{X}^T)^{-1} = \mathbf{I}_n - \mathbf{X}(\mathbf{I}_p/\gamma + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T.$$

The Schur complement condition discussed at length in [279] guarantees that as $\mathbf{I}_n + \gamma\mathbf{X}\mathbf{X}^T$ is strictly positive definite, we have the equivalence

$$2\eta \geq \mathbf{y}^T(\mathbf{I}_n + \gamma\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{y} \iff \begin{pmatrix} 2\eta & \mathbf{y}^T \\ \mathbf{y} & \mathbf{I}_n + \gamma\mathbf{X}\mathbf{X}^T \end{pmatrix} \in \mathbb{S}_+^{n+1}.$$

Representation (3.18) is thus an immediate consequence of expression (3.17) as well. \square

We next establish that the sparse regression problem (3.1) can in fact be represented as a pure binary optimization problem.

Theorem 3.2. *The sparse regression problem (3.1) can be reformulated as the nonlinear optimization problem*

$$\begin{aligned} \min \quad & \frac{1}{2}\mathbf{y}^T\left(\mathbf{I}_n + \gamma\sum_{j \in [p]} s_j \mathbf{K}_j\right)^{-1}\mathbf{y} \\ \text{s.t.} \quad & \mathbf{s} \in \mathbb{S}_k^p, \end{aligned} \tag{3.19}$$

where the kernel matrices \mathbf{K}_j in \mathbb{S}_+^n are defined as

$$\mathbf{K}_j = \mathbf{X}_j\mathbf{X}_j^T. \tag{3.20}$$

Proof. We start the proof by separating the optimization variable β in the sparse regression problem (3.1) into its support $s = \text{supp } \beta$ and the corresponding nonzero entries β_s . Evidently, we can now write the sparse regression problem (3.1) as the bilevel minimization problem

$$\min_{\mathbf{s} \in \mathbb{S}_k^p} \left[\min_{\beta_s \in \mathbb{R}^k} \frac{1}{2\gamma} \|\beta_s\|_2^2 + \frac{1}{2} \|\mathbf{y} - \mathbf{X}_s \beta_s\|_2^2 \right]. \tag{3.21}$$

It now remains to be shown that the inner minimum can be found explicitly as the objective function of the optimization problem (3.19). Using Lemma 3.1, the minimization problem can be reduced to the binary minimization problem $\min_{\mathbf{s} \in \mathbb{S}_k^p} c(\mathbf{X}_s \mathbf{X}_s^T)$. We finally remark that the outer product can be decomposed as the sum

$$\mathbf{X}_s \mathbf{X}_s^T = \sum_{j \in [p]} s_j \mathbf{X}_j \mathbf{X}_j^T,$$

thereby completing the proof. \square

The optimization problem (3.19) is a pure binary formulation of the sparse regression problem directly over the support s instead of the regressor β itself. As the objective function in (3.19) is convex in the vector \mathbf{s} , problem (3.19) casts the sparse regression problem as a convex integer optimization (CIO) problem. Nevertheless, we will never explicitly construct the CIO formulation as such and rather develop an efficient cutting plane algorithm. We finally discuss here how the sparse regression formulation in Theorem 3.2 is related to kernel regression and admits an interesting dual relaxation.

The kernel connection

In ordinary linear regression a linear relationship between input data \mathbf{X} and observations \mathbf{y} is determined through solving the least squares regression problem (3.15). The previous optimization problem is known as Ridge regression and balances the least-squares prediction error with a Tikhonov regularization term.

Formulation (3.2) represents a primal perspective on the sparse regression problem (3.1). Formulation (3.2) indeed attempts to solve the sparse regression problem in the primal space of parameters β directly.

However, it is well known in the kernel learning community that deeper results can be obtained if one approaches regression problems from its convex dual perspective due to [264]. Indeed, in most of the linear regression literature the dual perspective is often preferred over its primal counterpart. We state here the central result in this context to make the exposition self contained.

Theorem 3.3 ([264]). *The primal regression problem (3.15) can equivalently be formulated as the unconstrained maximization problem*

$$\begin{aligned} c = \max & -\frac{\gamma}{2} \boldsymbol{\alpha}^T \mathbf{K} \boldsymbol{\alpha} - \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{\alpha} + \mathbf{y}^T \boldsymbol{\alpha} \\ \text{s.t. } & \boldsymbol{\alpha} \in \mathbb{R}^n, \end{aligned} \tag{3.22}$$

where the kernel matrix $\mathbf{K} = \mathbf{X} \mathbf{X}^T$ in \mathbb{S}_+^n is a positive semidefinite matrix.

The dual optimization problem (3.22) is a convex QOP and, surprisingly, scales only with the number of samples n and is insensitive to the input dimension p . This last surprising observation is what gives the dual perspective its historical dominance over its primal counterpart in the context of kernelized regression discussed in [241]. When working with high dimensional data for which the number of inputs p is vastly bigger than the number of samples n , the dual optimization problem (3.22) is smaller and often easier to solve.

For any i and j , the kernel matrix entry $K(i, j)$ corresponds to the inner product between input samples \mathbf{x}_i and \mathbf{x}_j in \mathbb{R}^p . The matrix \mathbf{K} is usually referred to as the kernel matrix or Gram matrix and is

always positive definite and symmetric. Since the kernel specifies the inner products between all pairs of sample points in \mathbf{X} , it completely determines the relative positions of those points in the embedding space.

The CIO formulation (3.19) of the sparse regression problem (3.1) can be seen to take a dual perspective on the sparse regression problem (3.1). That is, optimization formulation (3.19) is recognized as a subset selection problem in the space of kernels instead of regressors. It can indeed be remarked that when the sparsity constraint is omitted the kernel matrix reduces to the standard kernel matrix

$$\mathbf{K} = \sum_{j \in [p]} \mathbf{X}_j \mathbf{X}_j^T = \mathbf{X} \mathbf{X}^T.$$

3.4 A cutting plane algorithm

We have formulated the sparse regression problem (3.1) as a pure binary convex optimization problem in Theorem 3.2. Unfortunately, no commercial solvers are available which are targeted to solve CIO problems of the type (3.19). In this section, we discuss a tailored solver. The algorithm is a cutting plane approach which iteratively solves increasingly better MIO approximations to the CIO formulation (3.19). Furthermore, the cutting plane algorithm avoids constructing the CIO formulation (3.19) explicitly which can prove burdensome when working with high-dimensional data. We provide numerical evidence in Section 3.5 that the algorithm described here is indeed extremely efficient.

Outer approximation algorithm

In order to solve the CIO problem (3.19), we construct a sequence of MIO approximations based on cutting planes. In pseudocode, it can be seen to construct a piece-wise affine lower bound to the convex regression loss function c defined in equation (3.18).

At each iteration, the cutting plane adds the inequality $\eta \geq c(\mathbf{s}_t) + \nabla c(\mathbf{s}_t)^T (\mathbf{s} - \mathbf{s}_t)$ cutting off the current binary solution \mathbf{s}_t unless \mathbf{s}_t happens to be optimal in (3.19). As the algorithm progresses, the outer approximation function c_t thus constructed

$$c_t(\mathbf{s}) = \max_{i \in [t]} c(\mathbf{s}_i) + \nabla c(\mathbf{s}_i)^T (\mathbf{s} - \mathbf{s}_i)$$

becomes an increasingly better approximation to the regression loss function c of interest. Unless the current binary solution \mathbf{s}_t is optimal, a new cutting plane will refine the feasible region of the problem by cutting off the current feasible binary solution.

Theorem 3.4 (Cutting Plane Method). *The procedure described in Algorithm 3.3 terminates after a finite number of cutting planes and returns the exact sparse regression solution β_0^* of (3.1).*

Algorithm 3.3 The outer approximation process.

Input: $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{X} \in \mathbb{R}^{n \times p}$ and $k \in [1, p]$.

Output: $\mathbf{s}_0^* \in \mathcal{S}_k^p$ and $\boldsymbol{\beta}_0^* \in \mathbb{R}^p$.

```

1: while  $\eta_t < c(\mathbf{s}_t)$  do
2:    $\mathbf{s}_{t+1}, \eta_{t+1} \leftarrow \begin{cases} \arg \min_{\mathbf{s}, \eta} \eta \\ \text{s.t. } \eta \geq c(\mathbf{s}_i) + \nabla c(\mathbf{s}_i)^T (\mathbf{s} - \mathbf{s}_i), \forall i \in [t], \\ \mathbf{s} \in \mathcal{S}_k^p, \end{cases}$ 
3:    $t \leftarrow t + 1$ 
4:    $\mathbf{s}_0^* \leftarrow \mathbf{s}_t$ 
5:    $\boldsymbol{\beta}_0^* \leftarrow 0$ 
6:    $\boldsymbol{\beta}_{\mathbf{s}_0^*}^* \leftarrow \left( \mathbf{I}_k / \gamma + \mathbf{X}_{\mathbf{s}_0^*}^T \mathbf{X}_{\mathbf{s}_0^*} \right)^{-1} \mathbf{X}_{\mathbf{s}_0^*}^T \mathbf{y}$ 

```

Despite the previous encouraging corollary of a result found in [99], it nevertheless remains the case that from a theoretical point of view exponentially many cutting planes need to be computed in the worst-case, potentially rendering the approach impractical. Furthermore, at each iteration a MIO problem needs to be solved. This can be done by constructing a branch-and-bound tree, which itself requires a potential exponential number of leaves to be explored. This complexity behavior is however to be expected as exact sparse regression is known to be an *NP*-hard problem. Surprisingly, the empirical timing results presented in Section 3.5 suggests that the situation is much more interesting than what complexity theory might suggest. In what remains of this section, we briefly discuss three techniques to carry out the outer approximation algorithm more efficiently than a naive implementation would.

In general, outer approximation methods are known as “multi-tree” methods because every time a cutting plane is added, a slightly different MIO problem is to be solved anew by constructing a branch-and-bound tree. Consecutive MIOs in Algorithm 3.3 differ only in one additional cutting plane. Over the course of the iterative cutting plane algorithm, a naive implementation would require that multiple branch and bound trees are built in order to solve the successive MIO problems. We implement a “single tree” way of solving the iteration Algorithm 3.3 by using dynamic constraint generation, known in the optimization literature as either a lazy constraint or column generation method. Lazy constraint formulations described in [9] dynamically add cutting planes to the model whenever a binary feasible solution is found. This saves the rework of rebuilding a new branch-and-bound tree every time a new binary solution is found in Algorithm 3.3. Lazy constraint callbacks are a relatively new type of callback.

In what follows, we discuss two additional tailored adjustments to the general outer approximation method which render the overall method more efficient. The first concerns an efficient way to evaluate both the regression

loss function c and its subgradient ∇c efficiently. The second discusses a heuristic to compute a warm start \mathbf{s}_1 to ensure that the first cutting plane added is of high quality, causing the outer approximation algorithm to converge more quickly.

Efficient dynamic constraint generation

Algorithm 3.3 adds linear constraints of the type

$$\eta \geq c(\bar{\mathbf{s}}) + \nabla c(\bar{\mathbf{s}})^T (\mathbf{s} - \bar{\mathbf{s}}) \quad (3.23)$$

at a given iterate $\bar{\mathbf{s}}$. We next evaluate both the regression loss function c and its subgradient components.

Lemma 3.2 (Derivatives of the optimal regression loss c). *Suppose the kernel matrix \mathbf{K} is differentiable at the parameter \mathbf{s} . Then, we have that the gradient of the regression loss function $c(\mathbf{K}) = \frac{1}{2}\alpha^*(\mathbf{K})^T \mathbf{y}$ can be stated as*

$$\nabla c(\mathbf{s}) = -\alpha^*(\mathbf{K})^T \cdot \frac{\gamma}{2} \frac{d\mathbf{K}}{d\mathbf{s}} \cdot \alpha^*(\mathbf{K}),$$

where $\alpha^*(\mathbf{K})$ maximizes (3.22) and hence is the solution to the linear system

$$\alpha^*(\mathbf{K}) = (\mathbf{I}_n + \gamma \mathbf{K})^{-1} \mathbf{y}.$$

We note that the naive numerical evaluation of the convex loss function c or any of its subgradients would require the inversion of the regularized kernel matrix $\mathbf{I}_n + \gamma \sum_{j \in [p]} \bar{s}_j \mathbf{K}_j$. The regularized kernel matrix is dense in general and always of full rank. Unfortunately, matrix inversion of general matrices presents work in the order of $\mathcal{O}(n^3)$ floating point operations and quickly becomes excessive for sample sizes n in the order of a few 1,000s. Bear in mind that such an inversion needs to take place for each cutting plane added in the outer approximation Algorithm 3.3.

It would thus appear that computation of the regression loss c based on its explicit characterization (3.17) is very demanding. Fortunately, the first explicit characterization (3.16) can be used to bring down the work necessary to $\mathcal{O}(k^3 + nk)$ floating point operations as we will show now. Comparing equalities (3.16) and (3.17) results immediately in the identity

$$\alpha^*(\sum_{j \in [p]} s_j \mathbf{K}_j) = (\mathbf{I}_n - \mathbf{X}_s (\mathbf{I}_k / \gamma + \mathbf{X}_s^T \mathbf{X}_s)^{-1} \mathbf{X}_s) \mathbf{y}. \quad (3.24)$$

The main advantage of the previous formula is the fact that it merely requires the inverse of the much smaller capacitance matrix $\mathbf{C} = \mathbf{I}_k / \gamma + \mathbf{X}_s^T \mathbf{X}_s$ in S_{++}^k instead of the dense full rank regularized kernel matrix in S_{++}^n .

Using expression (3.24), both the regression loss function c and any of its subgradients can be evaluated using $\mathcal{O}(k^3 + nk)$ instead of $\mathcal{O}(n^3)$ floating point operations. When the number of samples n is significantly larger than k , the matrix inversion lemma provides a significant edge over a vanilla matrix inversion.

3.5 Scalability and Phase Transitions

To evaluate the effectiveness of the cutting plane algorithm developed in Section 3.4, we report its ability to recover the correct regressors as well as its running time. In this section, we present empirical evidence on two critically important observations. The first observation is that the cutting plane algorithm scales to provable optimality in seconds for large regression problems with n and p in the 100,000s. This degree of scalability takes away the main propelling justification for heuristic approaches for many regression instances in practice. To evaluate the performance of an algorithm for the sparse regression problem, we define the accuracy and false alarm rate of a certain solution β^* in recovering the correct support as:

$$A\% = 100 \times \frac{|\text{supp}(\beta^{\text{true}}) \cap \text{supp}(\beta^*)|}{k}$$

and

$$F\% = 100 \times \frac{|\text{supp}(\beta^*) \setminus \text{supp}(\beta^{\text{true}})|}{|\text{supp}(\beta^*)|}.$$

Perfect support recovery occurs only then when β^* tells the whole truth ($A\% = 100$) and nothing but the truth ($F\% = 0$).

The second observation is that phase transition phenomena occur in the three important properties which characterize the dual approach to sparse regression: its ability to find all relevant features ($A\%$), its rejection of irrelevant features from the obfuscating bulk ($F\%$), and the time (T) it takes to find an exact sparse regressor using the cutting plane Algorithm 3.3.

All algorithms in this section are implemented in **Julia** and executed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running **CentOS release 6.7**. All optimization was done with the help of the commercial mathematical optimization distribution **Gurobi version 6.5**.

Synthetic Datasets

The input data samples $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ are drawn iid with

$$\mathbf{x}_i \sim N(\mathbf{0}, \Sigma), \quad \forall i = 1, \dots, n,$$

with $\sigma_{ij} = \rho^{|i-j|}$ for all $i, j = 1, \dots, p$. The noise components ϵ_i for $i = 1, \dots, n$ are drawn iid from a normal distribution $N(0, \sigma^2)$ and scaled to

$$\sqrt{\text{SNR}} = \|S\|_2 / \|\epsilon\|_2.$$

The unobserved true regressor β_{true} has exactly k -nonzero components at indices selected uniformly without replacement. Likewise, the nonzero coefficients in β_{true} are drawn uniformly at random from the set $\{-1, +1\}$. Finally, the response data \mathbf{y} are generated synthetically as

$$\mathbf{y} = \mathbf{X}\beta_{\text{true}} + \epsilon.$$

Table 3.3: A comparison between exact sparse regression using the cutting plane algorithm and **Lasso** with respect to their solution time in seconds applied to noisy ($\sqrt{\text{SNR}} = 20$) and lightly correlated data ($\rho = 0.1$) explained by either $k = 10$, $k = 20$ or $k = 30$ relevant features.

		Exact T for various n			Lasso T for various n			
		10k	20k	100k	10k	20k	100k	
k	10	$p = 50k$	21.2	34.4	310.4	69.5	140.1	431.3
		$p = 100k$	33.4	66.0	528.7	146.0	322.7	884.5
	κ	$p = 200k$	61.5	114.9	NA	279.7	566.9	NA
k	20	$p = 50k$	15.6	38.3	311.7	107.1	142.2	467.5
		$p = 100k$	29.2	62.7	525.0	216.7	332.5	988.0
	κ	$p = 200k$	55.3	130.6	NA	353.3	649.8	NA
k	30	$p = 50k$	31.4	52.0	306.4	99.4	220.2	475.5
		$p = 100k$	49.7	101.0	491.2	318.4	420.9	911.1
	κ	$p = 200k$	81.4	185.2	NA	480.3	884.0	NA

Scalability

In Table 3.3 we discuss the timing results for exact sparse linear regression as well as for **Lasso** applied to noisy ($\sqrt{\text{SNR}} = 20$) and lightly correlated ($\rho = 0.1$) synthetic data. We do not report the accuracy nor the false alarm rate of the obtained solution as this specific data is in the regime where exact discovery of the support occurs for both the **Lasso** and exact sparse regression.

The results in Table 3.3 provide evidence that Algorithm 3.3 represents a truly scalable algorithm to the exact sparse regression problem (3.1) for n and p in the 100,000s, and refute the widely held belief that exact sparse regression is not feasible at large scales.

Phase transition phenomena

We have established that the cutting plane Algorithm 3.3 scales to provable optimality for problems with n and p in the 100,000s. In the results presented in Table 3.3, both the exact and heuristic algorithms returned a sparse solution with correct support and otherwise were of similar precision. In cases where the data does not allow a statistically meaningful recovery of the ground truth β^{true} three phase transition phenomena occur. The first concerns the statistical power of sparse regression, whereas the second concerns our ability to find the optimal sparse regressor efficiently. We will refer to the former transition as the accuracy transition, while referring to the latter as the complexity transition. The false alarm phase transition is the third phase transition phenomenon and relates to the ability of exact sparse regression to reject irrelevant features from the obfuscating bulk.

We argue using strong empirical evidence that these transitions are in fact intimately related.

The accuracy phase transition describes the ability of the sparse regression formulation (3.1) to uncover the ground truth β^{true} from corrupted measurements alone. The corresponding phase transition for **Lasso** has been extensively studied in the literature by amongst many others [63, 128] and [266] and is considered well understood by now. Specifically, with uncorrelated input data ($\rho = 0$) [266] showed that for observations \mathbf{y} and independent Gaussian input data \mathbf{X} a phase transition occurs for **Lasso** for $n \geq n_1$ with

$$n_1 = (2k + \sigma^2) \log(p - k), \quad (3.25)$$

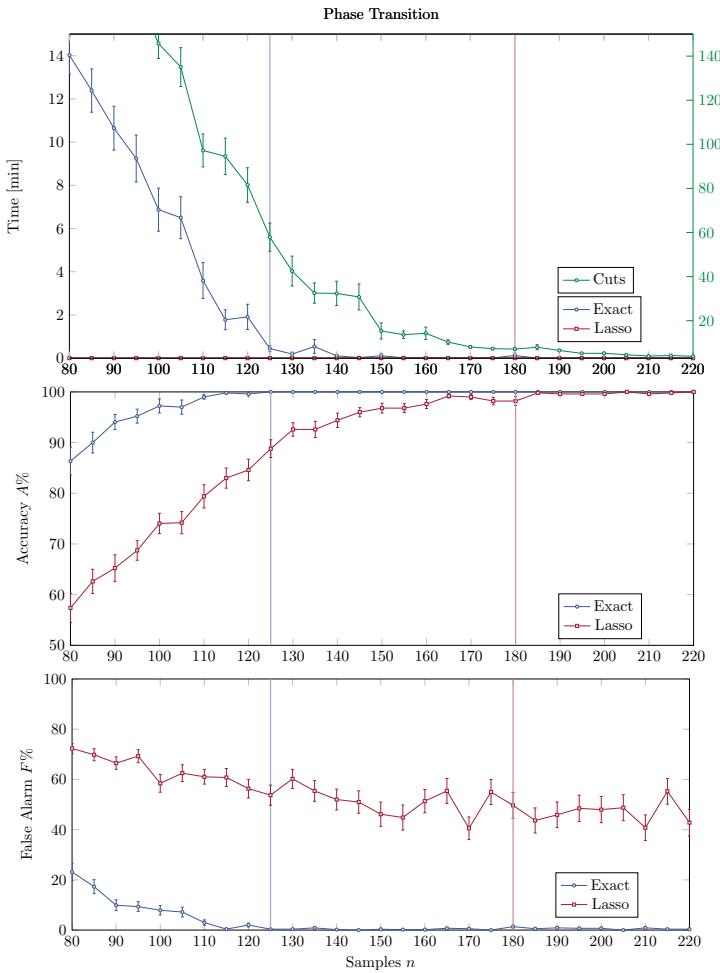
For sparse regression (3.1) with $\gamma = \infty$, independent Gaussian input data \mathbf{X} and $\beta^{\text{true}} \in \{0, 1\}^p$ [107] showed that a phase transition occurs for $n \geq n_0$ with

$$n_0 = 2k \log p / \log \left(\frac{2k}{\sigma^2} + 1 \right). \quad (3.26)$$

In Figure 3.6, we show empirical results for noiseless uncorrelated synthetically generated data with $p = 2,000$ of which only $k = 10$ are relevant. The accuracy $A\%$ and false alarm rates $F\%$ using exact sparse regression as well as the **Lasso** and time T in minutes to obtain either one are taken as the average values of fifty independent synthetic datasets. When the optimal solution is not found in less than fifteen minutes we take the best solution found up to that point. The error bars give an indication of one inter-sample standard deviation among these fifty independent experiments. The colored horizontal lines indicate that the number of samples n after which either method returned a full recovery ($A\% = 100$) of the support of the ground truth when both are given the correct number k of relevant sparse features. **Lasso** is empirically found to require approximately $n = 180$ samples to recover the true support which corresponds rather well with the theoretically predicted $n_1 = 152$ necessary samples by [266]. Unsurprisingly, the related accuracy phase transition of exact sparse regression using Algorithm 3.3 is found empirically to occur at $n_t = 125$ samples.

We now discuss the second transition which indicates that the time and the number of cuts it takes to solve the sparse regression (3.1) using the cutting plane Algorithm 3.3 experiences a phase transition as well. [35] show that that with high probability the number of cuts is merely one if $n > n_1$ and exact sparse regression is easy. Contrary to traditional complexity theory which suggests that the difficulty of a problem increases with problem size, the sparse regression problem has the property that as the number of samples $n > n_t > n_1$ increases the problem becomes easier in that the solution recovers 100% of the true signal, and the dual approach solves the problem extremely fast (in fact faster than **Lasso**), while for small number of samples $n < n_t$ exact sparse regression seems impractical.

Figure 3.6: A comparison between exact sparse regression using the cutting plane algorithm and **Lasso** on uncorrelated data ($\rho = 0$) with noise ($\sqrt{SNR} = 20$) counting $p = 2,000$ regressors of which only $k = 10$ are relevant. In the top panel we depict the time in minutes necessary to solve the sparse regression problem using either method as a function of the number of samples. The panel below gives the corresponding accuracy $A\%$ of the regressors as a function of the number of samples. The red vertical line at $n_1 = 152$ samples depicts the accuracy phase transition concerning the ability of the **Lasso** heuristic to recover the support of the ground truth w_{true} . The blue vertical line at $n_t = 125$ does the same for exact sparse regression. The final panel indicates the ability of both methods to reject obfuscating features in terms of the false alarm rate $F\%$. It can thus be seen that exact sparse regression does yields more statistically meaningful regressors (higher accuracy $A\%$ for less false alarms $F\%$) than **Lasso**. Furthermore, a complexity phase transition can be recognized as well all around n_t .



In all the experiments conducted up to this point, we assumed that the number of nonzero coefficients k of the ground truth β_{true} underlying the data was given. Evidently, in most practical applications the sparsity parameter k needs to be inferred from the data as well. In essence thus, any practical sparse regression procedure must pick those regressors contributing to the response out of the obfuscating bulk. To that end, we introduced the false alarm rate $F\%$ of a certain solution β^* as the percentage of regressors selected which are in fact unfitting. The ideal method would of course find all contributing regressors ($A\% = 100$) and not select any further ones ($F\% = 0$). In practice clearly, a trade-off must sometimes be made. The final phase transition will deal with the ability of exact sparse regression to reject irrelevant features using cross validation.

Historically, cross validation has been empirically found to be an effective way to infer the sparsity parameter k from data. Hence, for both exact sparse regression and Lasso, we select that number of nonzero coefficients which generalizes best to the validation sets constructed using cross validation with regards to prediction performance. In case of exact sparse regression, we let k range between one and twenty whereas the true unknown number of nonzero regressors was in fact ten. The third plot in Figure 3.6 gives the false alarm rate $F\%$ of both methods in terms of the number of samples n . As can be seen, Lasso has difficulty keeping a low false alarm rate with noisy data. Even in the region where Lasso is accurate ($A\%$), it is not as sparse as hoped for. Exact sparse regression does indeed yield sparser models as it avoids including regressors that do not contribute to the observations.

Parametric Dependency

To investigate the effect of each of the data parameters even further, we use synthetic data with the properties presented in Table 3.4. In order to be able to separate the effect of each parameter individually, we present the accuracy $A\%$, false alarm rate $F\%$ and solution time T of the cutting plane algorithm as a function of the number of samples n for each parameter value separately while keeping all other parameters fixed to their nominal value. All results are obtained as the average values of twenty independent experiments. The figures in the remainder of this section indicate that the accuracy, false alarm and complexity phase transitions shown in Figure 3.6 persist for a wide variety of properties of the synthetic data.

Feature dimension p

As both phase transition thresholds (3.25) and (3.26) depend only logarithmically on p , we do not expect the reported phase transitions to be very sensitive to p either. Indeed, in Figure 3.7 only a minor influence on the point of transition between statistically meaningful and efficient sparse

Table 3.4: Parameters describing the synthetic data used. The starred values denote the nominal values of each parameter.

Sparsity	k	$\{10^*, 15, 20\}$
Dimension	p	$\{5000^*, 10000, 15000\}$
Signal-to-noise ratio	$\sqrt{\text{SNR}}$	$\{3, 7, 20^*\}$

regression to unreliable and intractable regressors is observed as a function of p .

Sparsity level k

Figure 3.8 suggests that k has an important influence of the phase transition curve. The experiments suggest that there is a threshold f_t such that if $n/k \geq f_t$, then full support recovery ($A\% = 100$, $F\% = 0$) occurs and the time to solve problem (3.1) is in the order of seconds and only grows linear in n . Furthermore, if $n/k < f_t$, then support recovery $A\%$ drops to zero, false alarms $F\%$ surge, while the time to solve problem (3.1) grows combinatorially as $\binom{p}{k}$. This observation is in line with the theoretical result (3.26), which predicts that this threshold only depends logarithmically on the feature dimension p and the SNR which we study subsequently.

Signal-to-noise ratio (SNR)

From an information theoretic point of view, the SNR must play an important role as well as reflected by the theoretical curve (3.26). Indeed, the statistical power of any method is questionable when the noise exceeds the signal in the data. In Figure 3.9 this effect of noise is observed as for noisy data the phase transition occurs later than for more accurate data.

A remark on complexity

The empirical results in this section suggest that the traditional complexity point of view might be misleading towards a better understanding of the complexity of the sparse regression problem (3.1). For large n , sparse regression is easy to solve. Thus, contrary to traditional complexity theory which suggests that the difficulty of a problem increases with dimension, the sparse regression problem (3.1) has the property that for small number of samples n , Algorithm 3.3 takes a large amount of time to solve the problem. However, for a large number of samples n , the dual approach solves the problem extremely fast and recovers 100% of the support of the true regressor β_{true} .

Figure 3.7: The top panel shows the time it takes to solve the sparse regression problem using the cutting plane method for data with $p = 5,000, 10,000$ or $15,000$ regressors as a function of n . When the optimal solution is not found in less than ten minutes we take the best solution found up to that point. The bottom panels show the accuracy $A\%$ and false alarm rate $F\%$. Only a minor influence on the point of transition between statistically meaningful and efficient sparse regression to unreliable and intractable regression is observed as a function of the regression dimension p .

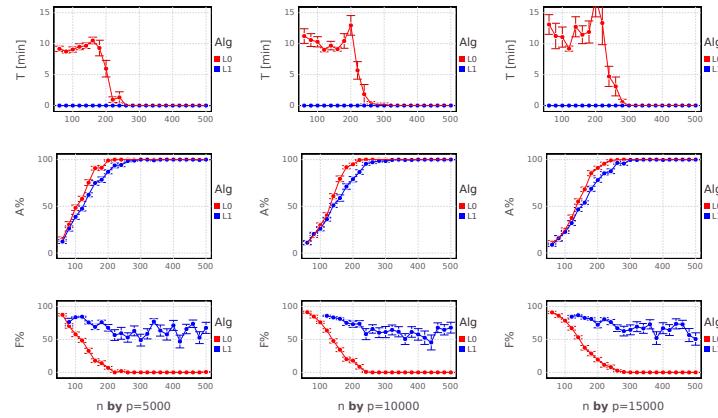


Figure 3.8: The top panel shows the time it takes to solve the sparse regression problem as a function of n using the cutting plane method for data with $p = 5,000$ regressors of which only $k = 10, 15$ or $k = 20$ are relevant. When the optimal solution is not found in less than ten minutes we take the best solution found up to that point. The bottom panels show the accuracy $A\%$ and false alarm rate $F\%$. These results suggest that the quantity n/k is a major factor in the phase transition curve of exact sparse regression.

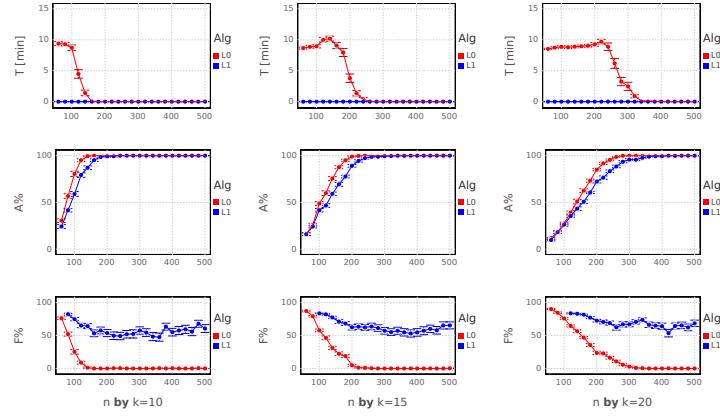
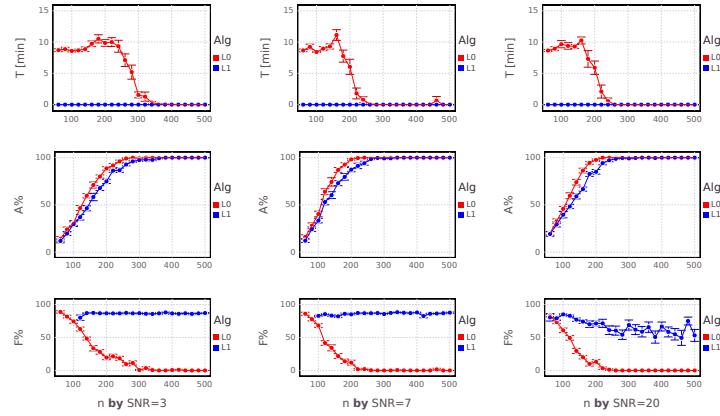


Figure 3.9: The top panel shows the time it takes to solve the sparse regression problem as a function of n using the cutting plane method for data with signal-to-noise level $\sqrt{\text{SNR}} = 3, 7$ and 20 . When the optimal solution is not found in less than one minute we take the best solution found up to that point. The bottom panel shows the accuracy $A\%$.



3.6 Concluding Remarks

We presented two approaches based on MIO for sparse linear regression. The primal approach is more direct and scales for n, p in the 1000s. In Chapter 5, we extend the primal approach to propose a holistic regression framework that models desirable properties for linear regression using MIO. The dual approach reformulates the problem as a CIO problem and scales to problems for n, p in the 100,000s. This level of scalability takes away the computational edge attributed to sparse regression heuristics such as the **Lasso** or **Elastic Net**.

The ability to solve sparse regression problems for very high dimensions allows us to observe new phase transition phenomena. We demonstrate experimentally that there is a threshold n_t such that if $n \geq n_t$, then β_0^* recovers the true support ($A\% = 100$ for $F\% = 0$) and the time to solve problem (3.1) is seconds (for n and p in 100,000s) and it only grows only linear in n . Remarkably, these times are less than the time to solve **Lasso** for similar sizes. Moreover, if $n < n_t$, then the time to solve problem (3.1) grows proportional to $\binom{p}{k}$. In other words, there is a phase transition in our ability to recover the true coefficients of the sparse regression problem and most surprisingly in our ability to solve it. Contrary to traditional complexity theory that suggests that the difficulty of a problem increases with dimension, the sparse regression problem (3.1) has the property that for small number of samples n , the dual approach takes a large amount of time to solve the problem, but most importantly the optimal solution does not recover the true signal. However, for a large number of samples n , the dual approach solves the problem extremely fast and recovers $A\% = 100$ of the support of the true regressor β_{true} . Significantly, the threshold n_t for the phase transition for full recovery of exact sparse regression is significantly smaller than the corresponding threshold n_1 for **Lasso**. Whereas **Lasso** tends to furthermore include many irrelevant features as well, exact sparse regression furthermore achieves this full recovery at almost $F\% = 0$ false alarm rate.

3.7 Notes and Sources

The primal approach to sparse regression described in Sections 3.1 and 3.2 is from [37]. The dual approach to sparse regression developed in Sections 3.3-3.5 is from [35].

Chapter 4

Nonlinear Regression

Truth is much too complicated to allow anything but approximations.

– John von Neumann

Contents

- 4.1. Convex Regression
- 4.2. Sparse Convex Regression
- 4.3. Median Regression
- 4.4. Concluding Remarks
- 4.5. Notes and Sources

In this chapter, we address natural generalizations of the linear regression problem. Given data $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, we wish to estimate

- (a) **Convex regression:** a convex function $f : \mathbb{R}^p \rightarrow \mathbb{R}$

$$\min_{f \in \mathcal{C}} \frac{1}{2} \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2, \quad (4.1)$$

where \mathcal{C} represents the space of convex functions on \mathbb{R}^p . Problem (4.1) is an optimization problem over functions.

- (b) **Sparse convex regression:** a convex function f with the property that the union of supports of the subgradients of f in each point \mathbf{x} is a set whose cardinality is bounded by k .

- (c) **Median regression:** a vector β

$$\min_{\beta} \left(\text{median}_{i=1,\dots,n} |y_i - \beta^T \mathbf{x}_i| \right), \quad (4.2)$$

and show that the convex and the MIO lenses lead to practable algorithms for these problems in high dimensions, which underscores the importance of the optimization lenses we use in this book.

4.1 Convex Regression

The convexity of f implies that for any pair $(\mathbf{x}_i, \mathbf{x}_j)$, the following conditions are necessary

$$f(\mathbf{x}_i) + \xi_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq f(\mathbf{x}_j),$$

with ξ_i being the subgradient of f at point \mathbf{x}_i . [51] show that it suffices to enforce these condition for all $n(n - 1)$ pairs of points $\mathbf{x}_i, \mathbf{x}_j$, $1 \leq i, j \leq n$. This implies that Problem (4.1) is equivalent to

$$\begin{aligned} & \min_{\theta, \{\xi_i\}_{i=1}^n} \frac{1}{2} \sum_{i=1}^n (y_i - \theta_i)^2 \\ \text{s.t. } & \theta_i + \xi_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j \quad \forall i, j, \\ & \theta \in \mathbb{R}^n, \\ & \xi_i \in \mathbb{R}^p \quad \forall i. \end{aligned} \quad (4.3)$$

The variables θ_i represent the values of $f(\mathbf{x}_i)$, and ξ_i belongs to the subdifferential set of the convex function f at each \mathbf{x}_i . This is a linear optimization problem with $n(n - 1)$ constraints, which can only be solved for small values of n . Instead, we develop a cutting plane based algorithm for solving Problem (4.3) that scales for n in 10^5 .

Cutting plane algorithms

To motivate the cutting plane algorithm for $p = 1$ we observe Problem (4.3) can be computed by solving with only $n - 1$ constraints, i.e., by sorting x_i 's and considering the adjacent pairs.

For $p > 1$, given $\mathbf{x}_1, \dots, \mathbf{x}_n$, we form a spanning path (SP) based on the Euclidean distances between these points. By starting from \mathbf{x}_{i_1} , we find its closest point \mathbf{x}_{i_2} based on the Euclidean distance and add it as the next point. We then find the closest point \mathbf{x}_{i_3} to \mathbf{x}_{i_2} over all the points excluding \mathbf{x}_{i_1} and \mathbf{x}_{i_2} , and so on. We utilize the $n - 1$ edges in the spanning path among $\mathbf{x}_1, \dots, \mathbf{x}_n$ as initial constraints. These $n - 1$ constraints initially form the reduced master problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}, \boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_n} \quad & \frac{1}{2} \|\mathbf{y} - \boldsymbol{\theta}\|^2 \\ \text{s.t.} \quad & \theta_{i_1} + \boldsymbol{\xi}'_{i_1} (\mathbf{x}_{i_2} - \mathbf{x}_{i_1}) \leq \theta_{i_2}, \\ & \theta_{i_2} + \boldsymbol{\xi}'_{i_2} (\mathbf{x}_{i_3} - \mathbf{x}_{i_2}) \leq \theta_{i_3}, \\ & \vdots \\ & \theta_{i_{n-1}} + \boldsymbol{\xi}'_{i_{n-1}} (\mathbf{x}_{i_n} - \mathbf{x}_{i_{n-1}}) \leq \theta_{i_n}, \end{aligned} \quad (4.4)$$

with the solution as $\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\xi}}_1, \dots, \hat{\boldsymbol{\xi}}_n$. Alternatively, we can compute the minimum spanning tree (MST) among $\mathbf{x}_1, \dots, \mathbf{x}_n$ and use the $n - 1$ edges of the MST as initial constraints or use randomly chosen pairs of points (Method (R)) as the initial reduced master problem or select the closest point for each point \mathbf{x}_i (Method (C)).

Delayed constraint generation

We check if a given solution $\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\xi}}_1, \dots, \hat{\boldsymbol{\xi}}_n$ to the reduced master problem (4.4) is feasible for the full problem by solving the so called separation problem

$$j(i) = \arg \max_{1 \leq k \leq n} \left\{ \hat{\theta}_i - \hat{\theta}_k + \hat{\boldsymbol{\xi}}'_i (\mathbf{x}_k - \mathbf{x}_i) \right\}, \quad (4.5)$$

and checking if the corresponding largest value is greater than 0. In practice, we only consider a constraint to be violated if it exceeds a given tolerance Tol . In the case of such a violation, we add the constraint

$$\theta_i + \boldsymbol{\xi}'_i (\mathbf{x}_{j(i)} - \mathbf{x}_i) \leq \theta_{j(i)} \quad (4.6)$$

to the reduced master problem for each i , and re-solve. Let T_k be the set index pairs of the violated constraints we add at the k^{th} iteration. Thus,

at the k^{th} iteration, the problem we solve is given by

$$\begin{aligned} \min_{\boldsymbol{\theta}, \boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_n} \quad & \frac{1}{2} \|\mathbf{y} - \boldsymbol{\theta}\|^2 \\ \text{s.t.} \quad & \theta_i + \boldsymbol{\xi}_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j, \forall (i, j) \in T_0, \\ & \theta_i + \boldsymbol{\xi}_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j, \forall (i, j) \in T_1, \\ & \vdots \\ & \theta_i + \boldsymbol{\xi}_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j, \forall (i, j) \in T_k. \end{aligned} \quad (4.7)$$

If $\max_{1 \leq k \leq n} \left\{ \hat{\theta}_i - \hat{\theta}_k + \hat{\boldsymbol{\xi}}'_i (\mathbf{x}_k - \mathbf{x}_i) \right\} \leq Tol \ \forall i \in [n]$, then the current solution is in fact optimal for the full problem (4.3) with $n(n-1)$ constraints, and the method terminates. The complete algorithm is as follows:

Algorithm 4.1 Cutting plane algorithm for Problem (4.3)

Input: Data (\mathbf{x}_i, y_i) , $i = 1, \dots, n$, tolerance $Tol > 0$.
Output: An optimal solution $(\boldsymbol{\theta}^*, \boldsymbol{\xi}_1^*, \dots, \boldsymbol{\xi}_n^*)$ to Problem (4.3).

- 1: Solve the reduced master problem, i.e., Problem (4.7) with $k = 0$.
- 2: Set $success = 0$.
- 3: **while** $success == 0$ **do**
- 4: **for** $1 \leq i \leq n$ **do**
- 5: For this i , solve the separation problem (4.5) to find a $j(i)$.
- 6: Add the corresponding violated constraint Eq. (4.6) to the reduced master problem.
- 7: If there is no violated constraint within the tolerance Tol , set $success \leftarrow 1$.
- 8: Else, re-solve Problem (4.7) with new constraint set T_{k+1} , consisting of additional constraint(s) added from Steps 4 – 7.
- 9: $k \leftarrow k + 1$

We also note that this cutting plane algorithm, by successively adding violated constraints to the reduced master problem, is guaranteed to converge to an optimal solution in a finite number of steps [158].

Algorithm 4.1 can be extended to accommodate the following additional requirements on $f(\cdot)$:

- (a) The function f is coordinate-wise monotone, i.e., its subgradients $\boldsymbol{\xi}_i$ are either $\boldsymbol{\xi}_i \geq \mathbf{0}$ or $\boldsymbol{\xi}_i \leq \mathbf{0}$ (non-decreasing or non-increasing respectively) for all i .
- (b) The subgradients $\boldsymbol{\xi}_i$ are bounded, i.e., $\|\boldsymbol{\xi}_i\|_p \leq L \ \forall i$ for some L and ℓ_p norm $\|\cdot\|_p$. The usual cases of $p \in \{1, 2, \infty\}$ result in conic optimization problems and can be handled by this approach. Such

constraints could be added as a part of the reduced master problem all at once, or in a delayed manner as and when they are violated.

[19] report computational results that show that the cutting plane method for the convex regression problem can be solved for sizes $(n, p) = (10^4, 10)$ in minutes and $(n, p) = (10^5, 10^2)$ in hours in both synthetic and real-world datasets.

4.2 Sparse Convex Regression

In this section, we consider the problem of sparse convex regression, in which the union of supports of the subgradients of f in each point x is a set whose cardinality is bounded by k . We formulate this as the following optimization problem over sets,

$$\begin{aligned} \min_{\boldsymbol{\theta}, \{\boldsymbol{\xi}_i\}_{i=1}^n, S} \quad & \frac{1}{2} \sum_{i=1}^n (y_i - \theta_i)^2 \\ \text{s.t.} \quad & \theta_i + \boldsymbol{\xi}_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j \quad \forall i, j, \\ & \text{Supp}(\boldsymbol{\xi}_i) \subseteq S \quad \forall i, \\ & \boldsymbol{\theta} \in \mathbb{R}^n, \\ & \boldsymbol{\xi}_i \in \mathbb{R}^p \quad \forall i, \\ & |S| \leq k, S \subseteq \{1, \dots, p\}. \end{aligned} \tag{4.8}$$

Primal approach

Consider the following MIQO problem

$$\begin{aligned} \min_{\boldsymbol{\theta}, \mathbf{z}, \{\boldsymbol{\xi}_i\}_{i=1}^n} \quad & \frac{1}{2} \sum_{i=1}^n (y_i - \theta_i)^2 \\ \text{s.t.} \quad & \theta_i + \boldsymbol{\xi}_i^T (\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j \quad \forall 1 \leq i, j \leq n, \\ & |(\boldsymbol{\xi}_i)_j| \leq M z_j \quad \forall i \in [n], j \in [p], \\ & \sum_{j=1}^p z_j \leq k, \\ & \mathbf{z} \in \{0, 1\}^p, \\ & \boldsymbol{\theta} \in \mathbb{R}^n, \\ & \boldsymbol{\xi}_i \in \mathbb{R}^p \quad \forall i \in [n]. \end{aligned} \tag{4.9}$$

for some positive constant M .

To solve this problem we first develop heuristics based on convex optimization which generate solutions and are fast in practice. We solve a reduced MIQO problem to generate lower bounds, which provide a guarantee on the quality of this solution.

Dual approach

We adapt the approach from Chapter 3 to this convex regression setting. We solve the following regularized problem, for a given $\gamma > 0$,

$$\begin{aligned} \min_{\boldsymbol{\theta}, \boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_n} \quad & \frac{1}{2} \|\mathbf{y} - \boldsymbol{\theta}\|^2 + \frac{1}{2\gamma} \sum_{i=1}^n \|\boldsymbol{\xi}_i\|^2 \\ \text{s.t.} \quad & \theta_i + \boldsymbol{\xi}'_i(\mathbf{x}_j - \mathbf{x}_i) \leq \theta_j \quad \forall i, j, \\ & \text{Supp}(\boldsymbol{\xi}_i) \subseteq S \quad \forall i, \\ & \boldsymbol{\theta} \in \mathbb{R}^n, \\ & \boldsymbol{\xi}_i \in \mathbb{R}^p \quad \forall i, \\ & |S| \leq k, S \subseteq \{1, \dots, p\}. \end{aligned} \tag{4.10}$$

We let

$$S_k^p = \{\mathbf{z} \in \{0, 1\}^p : \sum_{i=1}^p z_i \leq k\}.$$

and transform this problem to a binary optimization problem with a convex objective function.

Theorem 4.1. *Problem (4.10) is equivalent to solving the following binary optimization problem with convex objective, given by*

$$\min_{\mathbf{z} \in S_k^p} g(\mathbf{z}), \tag{4.11}$$

where

$$\begin{aligned} g(\mathbf{z}) = \max_{\boldsymbol{\mu} \geq \mathbf{0}} & -\frac{1}{2} \sum_{i=1}^n \left(y_i + \sum_{j=1}^n \mu_{ji} - \sum_{j=1}^n \mu_{ij} \right)^2 \\ & - \frac{\gamma}{2} \sum_{i=1}^n \sum_{\ell=1}^p z_\ell \left(\sum_{j=1}^n \mu_{ij} (x_{j\ell} - x_{i\ell}) \right)^2, \end{aligned} \tag{4.12}$$

and a subgradient of g is given by the vector with p^{th} element given by

$$(\partial g(\mathbf{z}))_\ell = -\frac{\gamma}{2} \sum_{i=1}^n \left(\sum_{j=1}^n \hat{\mu}_{ij} (x_{i\ell} - x_{j\ell}) \right)^2, \tag{4.13}$$

where $\hat{\boldsymbol{\mu}}$ is an optimal solution to the concave maximization problem given in Eq. (4.12).

We solve the outer problem as

$$\begin{aligned} \min_{\mathbf{z} \in \{0, 1\}^p, \eta} \quad & \eta \\ \text{s.t.} \quad & g(\mathbf{z}^i) + \partial g(\mathbf{z}^i)^T (\mathbf{z} - \mathbf{z}^i) \leq \eta, \quad i \in [m] \\ & \sum_{i=1}^p z_i \leq k, \end{aligned} \tag{4.14}$$

where m is the number of cuts added. We use dynamic constraint generation to avoid building multiple branch and bound trees. [19] report computational results that show that the dual method solves the sparse convex regression problem for sizes $(n, p, k) = (10^4, 10^2, 10)$ in minutes and $(n, p, k) = (10^5, 10^2, 10)$ in hours, while controlling for the false discovery rate effectively.

4.3 Median Regression

In this section, we solve

$$\min_{\beta} |r_{(q)}|, \quad (4.15)$$

where $r_i = |y_i - \beta^T \mathbf{x}_i|$ is the i -residual. For $q = n/2$, Problem (4.15) reduces to Problem (4.2).

We consider a list of the n residuals $|r_1|, \dots, |r_n|$, with the ordering

$$|r_{(1)}| \leq |r_{(2)}| \leq \dots \leq |r_{(n)}|.$$

To model the sorted q -th residual, i.e., $|r_{(q)}|$, we need to express the fact that $r_i \leq |r_{(q)}|$ for q many residuals $|r_i|$'s from $|r_1|, \dots, |r_n|$. To do so we introduce the binary variables $z_i, i = 1, \dots, n$ with the interpretation:

$$z_i = \begin{cases} 1, & \text{if } |r_i| \leq |r_{(q)}|, \\ 0, & \text{otherwise.} \end{cases}$$

We further introduce auxiliary continuous variables $\mu_i, \bar{\mu}_i \geq 0$ such that:

$$|r_i| - \mu_i \leq |r_{(q)}| \leq |r_i| + \bar{\mu}_i, \quad i = 1, \dots, n,$$

with the conditions:

$$\begin{array}{ll} \text{If } |r_i| \geq |r_{(q)}|, & \text{then } \bar{\mu}_i = 0, \mu_i \geq 0, \\ \text{and if } |r_i| \leq |r_{(q)}|, & \text{then } \mu_i = 0, \bar{\mu}_i \geq 0. \end{array} \quad (4.16)$$

We thus propose the following MIO formulation:

$$\begin{aligned} \min & \gamma \\ \text{s.t.} & |r_i| + \bar{\mu}_i \geq \gamma, \quad i = 1, \dots, n \\ & \gamma \geq |r_i| - \mu_i, \quad i = 1, \dots, n \\ & M_u z_i \geq \bar{\mu}_i, \quad i = 1, \dots, n \\ & M_\ell (1 - z_i) \geq \mu_i, \quad i = 1, \dots, n \\ & \sum_{i=1}^n z_i = q \\ & \mu_i \geq 0, \quad i = 1, \dots, n \\ & \bar{\mu}_i \geq 0, \quad i = 1, \dots, n \\ & z_i \in \{0, 1\}, \quad i = 1, \dots, n, \end{aligned} \quad (4.17)$$

where, $\gamma, z_i, \mu_i, \bar{\mu}_i, i = 1, \dots, n$ are the optimization variables, M_u, M_ℓ are sufficiently large constants. Let us denote the optimal solution of problem (4.17), which depends on M_ℓ, M_u , by γ^* . Suppose we consider $M_u, M_\ell \geq \max_i |r_{(i)}|$ —it follows from formulation (4.17) that q of the μ_i 's are zero. Thus, γ^* has to be larger than at least q of the $|r_i|$ values. By arguments similar to the above, we see that, since $(n - q)$ of the z_i 's are zero, at least $(n - q)$ many $\bar{\mu}_i$'s are zero. Thus γ^* is less than or equal to at least $(n - q)$ many of the $|r_i|, i = 1, \dots, n$ values. This shows that γ^* is indeed equal to $|r_{(q)}|$, for M_u, M_ℓ sufficiently large.

We found in our experiments that, in formulation (4.17), if $z_i = 1$, then $\bar{\mu}_i = M_u$ and if $z_i = 0$ then $\mu_i = M_\ell$. Though this does not interfere with the definition of $|r_{(q)}|$, it creates a difference in the strength of the MIO formulation. We describe below how to circumvent this shortcoming.

From (4.16) it is clear that $\bar{\mu}_i \mu_i = 0, \forall i = 1, \dots, n$. The constraint $\bar{\mu}_i \mu_i = 0$ can be modeled via integer optimization using Specially Ordered Sets of Type 1, i.e., SOS-1 constraints as follows:

$$\mu_i \bar{\mu}_i = 0 \iff (\mu_i, \bar{\mu}_i) : \text{SOS-1}, \quad (4.18)$$

for every $i = 1, \dots, n$. In addition, observe that, for M_ℓ sufficiently large and every $i \in \{1, \dots, n\}$ the constraint $M_\ell(1 - z_i) \geq \mu_i \geq 0$ can be modeled¹ by a SOS-1 constraint — $(\mu_i, z_i) : \text{SOS-1}$. In light of this discussion, we see that

$$|r_i| - |r_{(q)}| = \mu_i - \bar{\mu}_i, \quad (\mu_i, \bar{\mu}_i) : \text{SOS-1}. \quad (4.19)$$

We next show that $|r_{(q)}| \geq \bar{\mu}_i$ and $\mu_i \leq |r_i|$ for all $i = 1, \dots, p$. When $|r_i| \leq |r_{(q)}|$ it follows from the above representation that

$$\mu_i = 0 \text{ and } \bar{\mu}_i = |r_{(q)}| - |r_i| \leq |r_{(q)}|.$$

When $|r_i| > |r_{(q)}|$, it follows that $\bar{\mu}_i = 0$. Thus, it follows that $0 \leq \bar{\mu}_i \leq |r_{(q)}|$ for all $i = 1, \dots, n$. It also follows by a similar argument that $0 \leq \mu_i \leq |r_i|$ for all i .

Thus, by using SOS-1 type of constraints, we can avoid the use of Big-M's appearing in formulation (4.17), as follows:

$$\begin{aligned} \min \quad & \gamma \\ \text{s.t.} \quad & |r_i| - \gamma = \mu_i - \bar{\mu}_i, \quad i = 1, \dots, n \\ & \sum_{i=1}^n z_i = q \\ & \gamma \geq \bar{\mu}_i, \quad i = 1, \dots, n \\ & \bar{\mu}_i \geq 0, \quad i = 1, \dots, n \\ & \mu_i \geq 0, \quad i = 1, \dots, n \\ & (\bar{\mu}_i, \mu_i) : \text{SOS-1}, \quad i = 1, \dots, n \\ & (z_i, \mu_i) : \text{SOS-1}, \quad i = 1, \dots, n \\ & z_i \in \{0, 1\}, \quad i = 1, \dots, n. \end{aligned} \quad (4.20)$$

¹To see why this is true observe that $(\mu_i, z_i) : \text{SOS-1}$ is equivalent to $\mu_i z_i = 0$. Now, since $z_i \in \{0, 1\}$, we have the following possibilities: $z_i = 0$, in which case μ_i is free; if $z_i = 1$, then $\mu_i = 0$.

Note, however, that the constraints

$$|r_i| - \gamma = \mu_i - \bar{\mu}_i, \quad i = 1, \dots, n \quad (4.21)$$

are not convex in r_1, \dots, r_n . We thus introduce the following variables $r_i^+, r_i^-, i = 1, \dots, n$ such that

$$r_i^+ + r_i^- = |r_i|, \quad y_i - \mathbf{x}'_i \boldsymbol{\beta} = r_i^+ - r_i^-, \quad r_i^+ \geq 0, r_i^- \geq 0, \quad r_i^+ r_i^- = 0, \quad i = 1, \dots, n. \quad (4.22)$$

The constraint $r_i^+ r_i^- = 0$ can be modeled via SOS-1 constraints

$$(r_i^+, r_i^-) : \text{SOS-1 for every } i = 1, \dots, n.$$

This leads to the following MIO for Problem (4.15):

$$\begin{aligned} & \min \quad \gamma \\ \text{s.t.} \quad & r_i^+ + r_i^- - \gamma = \bar{\mu}_i - \mu_i, \quad i = 1, \dots, n \\ & r_i^+ - r_i^- = y_i - \mathbf{x}'_i \boldsymbol{\beta}, \quad i = 1, \dots, n \\ & \sum_{i=1}^n z_i = q \\ & \gamma \geq \mu_i \geq 0, \quad i = 1, \dots, n \\ & \mu_i \geq 0, \quad i = 1, \dots, n \\ & \bar{\mu}_i \geq 0, \quad i = 1, \dots, n \\ & r_i^+ \geq 0, r_i^- \geq 0, \quad i = 1, \dots, n \\ & (\bar{\mu}_i, \mu_i) : \text{SOS-1}, \quad i = 1, \dots, n \\ & (r_i^+, r_i^-) : \text{SOS-1}, \quad i = 1, \dots, n \\ & (z_i, \mu_i) : \text{SOS-1}, \quad i = 1, \dots, n \\ & z_i \in \{0, 1\}, \quad i = 1, \dots, n. \end{aligned} \quad (4.23)$$

Sequential Linear Optimization

We describe a sequential linear optimization approach to obtain a local minimum of Problem (4.15). We decompose the q th ordered absolute residual as follows:

$$|r_{(q)}| = |y_{(q)} - \mathbf{x}'_{(q)} \boldsymbol{\beta}| = \underbrace{\sum_{i=1}^{q+1} |y_{(i)} - \mathbf{x}'_{(i)} \boldsymbol{\beta}|}_{H_{q+1}(\boldsymbol{\beta})} - \underbrace{\sum_{i=1}^q |y_{(i)} - \mathbf{x}'_{(i)} \boldsymbol{\beta}|}_{H_q(\boldsymbol{\beta})}, \quad (4.24)$$

The function $H_m(\boldsymbol{\beta})$ can be written as

$$\begin{aligned} H_m(\boldsymbol{\beta}) := & \max_{\mathbf{w}} \quad \sum_{i=1}^n w_i |y_i - \mathbf{x}'_i \boldsymbol{\beta}| \\ \text{s.t.} \quad & \sum_{i=1}^n w_i = m \\ & 0 \leq w_i \leq 1, \quad i = 1, \dots, n. \end{aligned} \quad (4.25)$$

Let us denote the feasible set in problem (4.25) by

$$\mathcal{W}_m := \left\{ \mathbf{w} : \sum_{i=1}^n w_i = m, w_i \in [0, 1], \forall i = 1, \dots, n \right\}.$$

Observe that for every $\mathbf{w} \in \mathcal{W}_m$ the function $\sum_{i=1}^n w_i |y_i - \mathbf{x}'_i \boldsymbol{\beta}|$ is convex in $\boldsymbol{\beta}$. Furthermore, since $H_m(\boldsymbol{\beta})$ is the point-wise supremum with respect to \mathbf{w} over \mathcal{W}_m , the function $H_m(\boldsymbol{\beta})$ is convex in $\boldsymbol{\beta}$. By taking the dual of Problem (4.25) for $m = q + 1$ and invoking strong duality, we have:

$$\begin{aligned} H_{q+1}(\boldsymbol{\beta}) = & \min_{\theta, \boldsymbol{\nu}} \quad \theta(q+1) + \sum_{i=1}^n \nu_i \\ \text{s.t.} \quad & \theta + \nu_i \geq |y_i - \mathbf{x}'_i \boldsymbol{\beta}|, \quad i = 1, \dots, n \\ & \nu_i \geq 0, \quad i = 1, \dots, n. \end{aligned} \quad (4.26)$$

Hence, we have expressed the q th ordered absolute residual as a difference of two convex functions. Having expressed $H_{q+1}(\boldsymbol{\beta})$ as the value of a LO problem we linearize the function $H_q(\boldsymbol{\beta})$. Representation (4.25) also provides a characterization of the set of subgradients of $H_q(\boldsymbol{\beta})$:

$$\partial H_q(\boldsymbol{\beta}) = \text{conv} \left\{ \sum_{i=1}^n w_i^* \text{sign}(y_i - \mathbf{x}'_i \boldsymbol{\beta}) \mathbf{x}_i : \mathbf{w}^* \in \arg \max_{\mathbf{w} \in \mathcal{W}_q} \mathcal{L}(\boldsymbol{\beta}, \mathbf{w}) \right\}, \quad (4.27)$$

where $\mathcal{L}(\boldsymbol{\beta}, \mathbf{w}) = \sum_{i=1}^n w_i |y_i - \mathbf{x}'_i \boldsymbol{\beta}|$ and $\text{conv}(S)$ denotes the convex hull of set S . An element of the set of subgradients (4.27) will be denoted by $\partial H(\boldsymbol{\beta})$.

If $\boldsymbol{\beta}_k$ denotes the value of the estimate at iteration k , we linearize $H_q(\boldsymbol{\beta})$ at $\boldsymbol{\beta}_k$ as follows:

$$H_q(\boldsymbol{\beta}) \approx H_q(\boldsymbol{\beta}_k) + \langle \partial H_q(\boldsymbol{\beta}_k), \boldsymbol{\beta} - \boldsymbol{\beta}_k \rangle. \quad (4.28)$$

Combining (4.26) and (4.28) we obtain that minimizing (4.24) with respect to $\boldsymbol{\beta}$ can be approximately done by solving the following LO problem:

$$\begin{aligned} \min_{\boldsymbol{\nu}, \theta, \boldsymbol{\beta}} \quad & \theta(q+1) + \sum_{i=1}^n \nu_i - \langle \partial H_q(\boldsymbol{\beta}_k), \boldsymbol{\beta} \rangle \\ \text{s.t.} \quad & \theta + \nu_i \geq |y_i - \mathbf{x}'_i \boldsymbol{\beta}|, \quad i = 1, \dots, n \\ & \nu_i \geq 0, \quad i = 1, \dots, n. \end{aligned} \quad (4.29)$$

Let $\boldsymbol{\beta}_{k+1}$ denote a minimizer of Problem (4.29). This leads to an iterative optimization procedure as described in Algorithm 4.2.

Algorithm 4.2 converges to a first order stationary point of (4.15), see [33].

Algorithm 4.2 Sequential Linear Optimization Algorithm for Problem (4.15)

Input: Data (\mathbf{x}_i, y_i) , $i = 1, \dots, n$, q , tolerance $Tol > 0$.

Output: An optimal solution β to Problem (4.15).

- 1 Initialize with β_1 , and for $k \geq 1$ perform the following Steps 2-3 for predefined tolerance parameter “Tol”.
 - 2 Solve the linear optimization problem (4.29) and let $(\nu_{k+1}, \theta_{k+1}, \beta_{k+1})$ denote a minimizer.
 - 3 If $(|y_{(q)} - \mathbf{x}_{(q)}^T \beta_k| - |y_{(q)} - \mathbf{x}_{(q)}^T \beta_{k+1}|) \leq Tol \cdot |y_{(q)} - \mathbf{x}_{(q)}^T \beta_k|$ exit; else go to Step 2.
-

A First-order Subgradient Method

Subgradient descent methods have a long history in non-smooth convex optimization. If computation of the subgradients turns out to be inexpensive, then subgradient based methods are quite effective in obtaining a moderate accuracy solution with relatively low computational cost. For non-convex and non-smooth functions, a subgradient need not exist, so the notion of a subgradient needs to be generalized. For non-convex, non-smooth functions having certain regularity properties (for example, Lipschitz functions) subdifferentials exist and form a natural generalization of subgradients. In particular, the quantity:

$$\partial f_q(\beta) = -\text{sign}(y_{(q)} - \mathbf{x}_{(q)}^T \beta) \mathbf{x}_{(q)}$$

is a subdifferential of the function $f_q(\beta) = |r_{(q)}|$ at β . We next present Algorithm 4.3 for Problem (4.15).

Algorithm 4.3 Subgradient Descent Algorithm for the LQS problem

Input: Data (\mathbf{x}_i, y_i) , $i = 1, \dots, n$, q , maximum number of iterations MaxIter.

Output: An optimal solution β to Problem (4.15).

1. Initialize β_1 , for $\text{MaxIter} \geq k \geq 1$ do the following:
 2. $\beta_{k+1} = \beta_k - \alpha_k \partial f_q(\beta_k)$ where α_k is a step-size.
 3. Return $\min_{1 \leq k \leq \text{MaxIter}} f_q(\beta_k)$ and β_{k*} at which the minimum is attained, where $k^* = \arg \min_{1 \leq k \leq \text{MaxIter}} f_q(\beta_k)$.
-

While various step-size choices are possible, we found the following

simple fixed step-size sequence to be quite useful in our experiments:

$$\alpha_k = \frac{1}{\max_{i=1,\dots,n} \|\mathbf{x}_i\|_2},$$

the above quantity $\max_{i=1,\dots,n} \|\mathbf{x}_i\|_2$ may be interpreted as an upper bound to the subdifferentials of $f_q(\boldsymbol{\beta})$.

Scalability to Large Problems

For large scale problems with $n \geq 5000$ with $p \geq 10$, we found that Algorithm 4.2 becomes computationally demanding due to the associated LO problems (4.29) appearing in Step-2 of Algorithm 4.2. On the other hand, Algorithm 4.3 remains computationally inexpensive. For larger problems, we run Algorithm 4.3 for several random initializations and find the best solution among them. [33] report that the best solution is found by the MIO with warm starts generated by Algorithm 4.3 in about 1.5 hours. Algorithm 4.3 found a solution within 1% of the MIO solution within minutes.

4.4 Concluding Remarks

In this chapter, we showed how the convex and MIO lenses leads to near optimal solutions to natural nonlinear extensions to linear regression. A combination of cutting planes, the dual method we saw in Chapter 3, MIO formulations and first-order methods used creatively and in combination provide a strong arsenal that can help tackle nonlinear regression problems.

4.5 Notes and Sources

Sections 4.1 and 4.2 are from [19]. Section 4.3 is from [33].

Chapter 5

Holistic Regression

The whole is more than the sum of its parts.

– Aristotle

Contents

- 5.1. Desirable Properties of a Linear Regression Model
- 5.2. Low Global Multicollinearity
- 5.3. Examples of Holistic Regression
- 5.4. Insights from Computations
- 5.5. Concluding Remarks
- 5.6. Notes and Sources

Linear regression models are traditionally built through trial and error in order to balance many competing goals such as predictive power, sparsity, interpretability, significance, robustness to errors in data, and lack of multi-collinearity, among others. These properties of a high quality linear regression model are typically built into the model one at a time and through repeated trial and error by the modeler. Hence, there is no guarantee that the final model addresses all of them satisfactorily, let alone optimally. In this chapter, we propose a holistic regression framework, a discrete optimization approach which simultaneously takes into account these desirable properties and, whenever it is not possible to satisfy all these properties simultaneously, the optimization model provides a guarantee that it is indeed not possible to do so.

Humans and machines have different strengths and the approach in this chapter aims to utilize both these strengths. The modeler typically has subject matter expertise; for example, the modeler may know of a particular structure present in the data, or can require that certain variables be present in the final model. While humans have intuition and contextual knowledge and understanding, computers have significantly more computational power. The objective of this chapter is to empower modelers with a methodology that builds models with properties that a human modeler can require based on intuition and expertise.

5.1 Desirable Properties of a Linear Regression Model

In this section, we review desirable characteristics of a linear regression model with response vector $\mathbf{y}_{n \times 1}$, model matrix $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_p] \in \mathbb{R}^{n \times p}$, regression coefficients $\boldsymbol{\beta} \in \mathbb{R}^{p \times 1}$ and errors $\boldsymbol{\epsilon} \in \mathbb{R}^{n \times 1}$:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}.$$

We introduce decision variable z_j that takes value 1, if variable x_j , is selected, and 0, otherwise, $j = 1, \dots, p$.

Sparsity

As we discussed in Chapter 3 when the number of potential features is large, we often wish to identify a critical subset which is primarily responsible for producing the response. This leads to more interpretable models, and aids prediction accuracy by eliminating noise variables to increase the model's ability to generalize. For this reason, we want to develop linear regression models with a specified number k of nonzero coefficients β . As we have seen in Chapter 3, we can control sparsity using the variables z_j , $j = 1, \dots, p$ as follows:

$$-Mz_j \leq \beta_j \leq Mz_j, \quad i = 1, \dots, p,$$

$$\sum_{j=1}^p z_j \leq k,$$

where M is a large number.

Group Sparsity

Some applications exhibit a group-sparse structure, with groups of independent variables whose coefficients are either all zero or all nonzero. Categorical variables, when expressed as a collection of dummy variables, form a natural group structure. Clear group formations also appear in compressed sensing ([92]), microarray analysis ([184]), and other applications. Specifically, we are given groups of variables GS_m , $m = 1, \dots, G$ and we require that among the variables GS_m are either all selected or none is selected. This can be accomplished by imposing the constraints

$$z_i = z_j, \quad i, j \in GS_m, \quad m = 1, \dots, G.$$

Limited Pairwise Multicollinearity

A near-linear relationship between independent variables obfuscates the relationship of each feature to the response and leads to unstable parameter estimates. In the worst case, multicollinearity can even cause parameter estimates to take the opposite sign from their true contribution to the response. To avoid these issues and produce interpretable models, a high quality regression model will contain features that are as orthogonal as possible. Let \mathcal{HC} be the set of pairs of variables with correlation at least ρ . To avoid pairwise multicollinearity we impose the constraints

$$z_i + z_j \leq 1, \quad (i, j) \in \mathcal{HC},$$

that is we limit the independent variables in the regression model to those which have pairwise correlation less than ρ .

Detecting Appropriate Nonlinear Transformations

As we discussed in Chapter 4 the data may not be collected in the units that are most explanatory of the dependent variable. It may turn out that a nonlinear transformation of an independent variable results in a new variable which can explain the variance in the dependent variable much better than the original measured variable could. For any variable j for which nonlinear transformations may be desired, we include all the potential transformed versions of the variable in the dataset. Let the set T_j contain the original variable x_j and its nonlinear transformations, for example x_j^2 , $\log x_j$, etc. We impose the constraint

$$\sum_{i \in T_j} z_i \leq 1, \quad j = 1, \dots, p,$$

which implies that among all nonlinear transformations for variable x_j , we only select one.

Robustness

As discussed in Section 2, it is very common that the data used in regression models are inaccurate. We have seen that if U represents the uncertainty set for the error $\Delta\mathbf{X}$ with

$$U = \{\Delta\mathbf{X} \mid \|\Delta\mathbf{X}\|_{r,q} \leq \Gamma\}, \text{ where } \|\mathbf{A}\|_{\ell,q} = \max_{\|\mathbf{z}\|_q=1} \|\mathbf{Az}\|_\ell.$$

then

$$\min_{\beta} \max_{\Delta\mathbf{X} \in U} \frac{1}{2} \|\mathbf{y} - (\mathbf{X} + \Delta\mathbf{X})\beta\|_\ell^\ell = \min_{\beta} \frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_\ell^\ell + \Gamma \|\beta\|_q.$$

Using $\ell = 2$ and $q = 1$, we thus impose robustness by regularizing the objective function to

$$\min_{\beta, \mathbf{z}} \frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \Gamma \|\beta\|_1.$$

Statistical Significance

Statistical inference relies not just on parameter estimates, but on specifications of uncertainty and confidence regarding those estimates. It is critical when interpreting parameters to have a sense of whether the model is truly detecting an underlying relationship between the variable and the response. The standard way of quantifying this in the scientific literature is through the concept of statistical significance. An independent variable in a regression model is labeled as “statistically significant” if, in the presence of the other variables in the model, the probability α that the observed effect occurred by chance is low, conventionally 5% or less. Modelers typically exclude insignificant variables from regression models, because they can only give murky interpretations of their effects on the response.

We would like holistic regression to select variables that are statistically significant, independent of distributional assumptions. For this reason we extend a standard result about the asymptotic guarantee of the t -test statistic to serve as the basis of holistic regression. For a linear regression problem:

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon,$$

when $\epsilon \sim N(0, \sigma^2 \mathbf{I})$,

$$\frac{\hat{\beta}_j - \beta_j}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} \sim t_{n-p}. \quad (5.1)$$

Here $\hat{\beta}_j$ is the least squares estimate of β_j with

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}.$$

Let $\mathbf{K} = (\mathbf{X}^T \mathbf{X})^{-1}$ and

$$\tilde{\sigma} = \sqrt{\frac{\mathbf{Y}^T (\mathbf{I}_n - \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T) \mathbf{Y}}{n-p}}$$

is the least squares estimate of standard deviation σ and t_{n-p} is the Student t -distribution with $n-p$ degrees of freedom.

We next extend (5.1) by removing the normality assumption for ϵ . We define

$$M_{jk} = \sum_{i=1}^p K_{ji} X_{ki} Y_k.$$

Taking expectations with respect to ϵ , we have

$$E[M_{jk}] = \sum_{m=1}^p \sum_{i=1}^p K_{ji} X_{ki} X_{km} \beta_m \quad \text{Var}(M_{jk}) = \sigma^2 \cdot \sum_{i=1}^p K_{ji}^2 X_{ki}^2.$$

Theorem 5.1. *We assume that for each $j = 1, \dots, p$, there exists a $\delta > 0$ such that*

$$\lim_{n \rightarrow \infty} \frac{1}{\left(\sqrt{\text{Var}(M_{jk})}\right)^{2+\delta}} \sum_{j=1}^n E[|M_{jk} - E[M_{jk}]|^{2+\delta}] = 0.$$

Then, for fixed p , we have:

$$\frac{\hat{\beta}_j - \beta_j}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} \xrightarrow{d} t_{n-p}$$

as $n \rightarrow \infty$.

This theorem shows that under mild regularity conditions of the error function, asymptotically the t -test is still valid, even if the error terms are not normal.

For a test of size α , we first define the quantity $t_\alpha = t_p^{-1}(1 - \frac{\alpha}{2})$, the inverse cdf of the t_p distribution at point $1 - \frac{\alpha}{2}$. Then, we can impose the t -test by requiring:

$$\frac{|\beta_j|}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} \geq t_\alpha z_j,$$

which is equivalent to the big M -constraints:

$$\begin{aligned} \frac{\beta_j}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} + Mb_j &\geq t_\alpha z_j \\ -\frac{\beta_j}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} + M(1 - b_j) &\geq t_\alpha z_j \\ b_j \in \{0, 1\}, \quad j = 1, \dots, p, \end{aligned}$$

where M is a large constant.

5.2 Low Global Multicollinearity

We note that it is possible to have multicollinearity without having any high pairwise correlations. Thus, using a pairwise correlation threshold as a surrogate for eliminating multicollinearity may not catch all cases of multicollinearity. Given data \mathbf{X} , we would like the design matrix to be free of multicollinear relationships so that $\det(\mathbf{X}^T \mathbf{X})$ is not very close to 0. We denote the columns of \mathbf{X} as \mathbf{X}_j , $j = 1, \dots, p$.

We introduce the vector $(1, \dots, 1)^T$ into the design matrix as a new column (the intercept) and we can define the multicollinear relationship as:

Definition 2. A set of variables $\mathbf{X}_1, \dots, \mathbf{X}_p$ has an ϵ -multicollinear relationship if for some $\mathbf{a} \in \mathbb{R}^p$, $\|\mathbf{a}\| = 1$, we have that:

$$\left\| \sum_{j=1}^n a_j \mathbf{X}_j \right\| < \epsilon. \quad (5.2)$$

We first establish the key result that connects the existence of an ϵ -multicollinear relationship (5.2) to the existence of an eigenvector \mathbf{v} for the matrix $\mathbf{X}^T \mathbf{X}$ that has a small ($O(\sqrt{\epsilon})$) eigenvalue. We then find multicollinear relations ($\mathbf{a} = (a_1, \dots, a_p)$) using information from the small eigenvalues of the matrix $\mathbf{X}^T \mathbf{X}$. We introduce the idea of a *minimum-support multicollinear relationship*. We finally propose an algorithm that uses the theory from the previous steps to identify all the multicollinear relationships.

Key Result

In this section, we establish a connection between the existence of an ϵ -multicollinear relationship and the existence of a eigenvector \mathbf{v} for the matrix $\mathbf{X}^T \mathbf{X}$ with a small ($O(\sqrt{\epsilon})$) eigenvalue:

Theorem 5.2. Let $V = \{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ be the set of orthonormal eigenvectors of $\mathbf{X}^T \mathbf{X} \in \mathbb{R}^{p \times p}$ such that the eigenvalues associated with V are less than ϵ . Then for $\mathbf{a} \in \mathbb{R}^p$, $\|\mathbf{a}\| = 1$:

- (a) If $\left\| \sum_{j=1}^p a_j \mathbf{X}_j \right\| < \epsilon$, then there exists a vector $\mathbf{b} \in \mathbb{R}^p$, $\|\mathbf{b}\| < (p - m)\sqrt{\epsilon}$ such that $\mathbf{a} - \mathbf{b} \in \text{Span}(V)$.
- (b) If there exists a vector $\mathbf{b} \in \mathbb{R}^p$, $\|\mathbf{b}\| < \sqrt{\epsilon}$ such that $\mathbf{a} - \mathbf{b} \in \text{Span}(V)$, then we have:

$$\left\| \sum_{j=1}^p a_j \mathbf{X}_j \right\| < \sqrt{(1 + \lambda_{m+1} + \dots + \lambda_p)\epsilon},$$

where $\lambda_{m+1}, \dots, \lambda_p$ are the eigenvalues associated with the set of orthonormal eigenvectors of M that have value greater or equal to ϵ .

Theorem 5.2 represents a weak equivalence between a small multicollinear relationship and the existence of a vector \mathbf{a} that is close to $\text{Span}(V)$, in the sense that there exists a small vector \mathbf{b} with $\|\mathbf{b}\| < O(\sqrt{\epsilon})$ such that $\mathbf{a} - \mathbf{b} \in \text{Span}(V)$. The proof is as follows:

Proof.

- (a) If $m = p$, then every $\mathbf{a} \in \text{Span}(V)$. Thus, we assume $m < p$ and prove part (a) by contradiction. We assume there exists no $\mathbf{b} \in \mathbb{R}^p$ with $\|\mathbf{b}\| < (p-m)\sqrt{\epsilon}$ such that $\mathbf{a} - \mathbf{b} \in \text{Span}(V)$. Let $\lambda_1, \dots, \lambda_p$ be the corresponding eigenvalues to eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_p$. Note that we have $0 \leq \lambda_1, \dots, \lambda_m < \epsilon$, and $\epsilon \leq \lambda_{m+1}, \dots, \lambda_p$. We write \mathbf{a} as:

$$\mathbf{a} = \alpha_1 \mathbf{v}_1 + \dots + \alpha_p \mathbf{v}_p.$$

Letting $\mathbf{b} = \alpha_{m+1} \mathbf{v}_{m+1} + \dots + \alpha_p \mathbf{v}_p$, we have that $\mathbf{a} - \mathbf{b} \in \text{Span}(V)$ by construction, which implies that:

$$\|\mathbf{b}\| = \|\alpha_{m+1} \mathbf{v}_{m+1} + \dots + \alpha_p \mathbf{v}_p\| \geq (p-m)\sqrt{\epsilon}.$$

This implies that there exists a α_{j_0} , $j_0 \in \{m+1, \dots, p\}$ such that $\|\alpha_{j_0}\| \geq \sqrt{\epsilon}$. Now,

$$\left\| \sum_{j=1}^p a_j \mathbf{X}_j \right\| = \|\mathbf{X}\mathbf{a}\| = \|\alpha_1 \mathbf{X}\mathbf{v}_1 + \dots + \alpha_p \mathbf{X}\mathbf{v}_p\| < \epsilon.$$

We have

$$\begin{aligned} \epsilon^2 &> \mathbf{a}^T \mathbf{X}^T \mathbf{X} \mathbf{a} \\ &= (\alpha_1 \mathbf{X}\mathbf{v}_1 + \dots + \alpha_p \mathbf{X}\mathbf{v}_p)^T (\alpha_1 \mathbf{X}\mathbf{v}_1 + \dots + \alpha_p \mathbf{X}\mathbf{v}_p) \\ &= \alpha_1^2 \lambda_1 + \dots + \alpha_p^2 \lambda_p \\ &\geq \alpha_{j_0}^2 \lambda_{j_0}. \end{aligned}$$

Since $|\alpha_{j_0}| \geq \sqrt{\epsilon}$, and $\lambda_{j_0} \geq \epsilon$, we have that $\epsilon^2 > \alpha_{j_0}^2 \lambda_{j_0} \geq \epsilon^2$, a contradiction.

- (b) If $m = p$, then $\mathbf{a} = \sum_{j=1}^p a_j \mathbf{v}_j$. Note $\|\mathbf{a}\|^2 = \sum_{j=1}^p a_j^2$, since \mathbf{v}_j are orthonormal. Hence, for $\|\mathbf{a}\| = 1$

$$\begin{aligned} \|\mathbf{X}\mathbf{a}\|^2 &= \mathbf{a}^T \mathbf{X}^T \mathbf{X} \mathbf{a} \\ &= \left(\sum_{j=1}^p a_j \mathbf{v}_j \right)^T \mathbf{X}^T \mathbf{X} \left(\sum_{j=1}^p a_j \mathbf{v}_j \right) \\ &= \left(\sum_{j=1}^p a_j \mathbf{v}_j \right)^T \left(\sum_{j=1}^p a_j \lambda_j \mathbf{v}_j \right) \\ &= \sum_{j=1}^p \lambda_j a_j^2 \\ &< \epsilon \|\mathbf{a}\|^2 = \epsilon. \end{aligned}$$

leading to $\|\mathbf{X}\mathbf{a}\| < \sqrt{\epsilon}$. We assume $m < p$. We write \mathbf{a} as:

$$\mathbf{a} = \alpha_1 \mathbf{v}_1 + \dots + \alpha_p \mathbf{v}_p$$

and observe that:

$$\min_{\mathbf{u} \in \text{Span}(V)} \|\mathbf{a} - \mathbf{u}\| = \|\alpha_{m+1} \mathbf{v}_{m+1} + \dots + \alpha_p \mathbf{v}_p\|. \quad (5.3)$$

Since by assumption there exists a \mathbf{b} with $\|\mathbf{b}\| < \sqrt{\epsilon}$ and $\mathbf{a} - \mathbf{b} \in \text{Span}(V)$, the vector $\mathbf{a} - \mathbf{b}$ is a feasible solution to problem (5.3), and thus taking $\mathbf{u} = \mathbf{a} - \mathbf{b}$ we have:

$$\|\mathbf{a} - (\mathbf{a} - \mathbf{b})\| = \|\mathbf{b}\| \geq \|\alpha_{m+1} \mathbf{v}_{m+1} + \dots + \alpha_p \mathbf{v}_p\|,$$

leading to:

$$\|\alpha_{m+1} \mathbf{v}_{m+1} + \dots + \alpha_p \mathbf{v}_p\| < \sqrt{\epsilon}.$$

Since

$$\|\alpha_{m+1} \mathbf{v}_{m+1} + \dots + \alpha_p \mathbf{v}_p\|^2 = \sum_{j=m+1}^p \alpha_j^2 < \epsilon,$$

we obtain that $|\alpha_j| < \sqrt{\epsilon}$ for all $j = m+1, \dots, p$. Thus, we have:

$$\begin{aligned} \mathbf{a}^T \mathbf{X}^T \mathbf{X} \mathbf{a} &= (\alpha_1 \mathbf{X} \mathbf{v}_1 + \dots + \alpha_p \mathbf{X} \mathbf{v}_p)^T (\alpha_1 \mathbf{X} \mathbf{v}_1 + \dots + \alpha_p \mathbf{X} \mathbf{v}_p) \\ &= \alpha_1^2 \lambda_1 + \dots + \alpha_p^2 \lambda_p \\ &= (\alpha_1^2 \lambda_1 + \dots + \alpha_m^2 \lambda_m) + (\alpha_{m+1}^2 \lambda_{m+1} + \dots + \alpha_p^2 \lambda_p) \\ &\leq \epsilon \cdot \sum_{j=1}^m \alpha_j^2 + \epsilon \cdot \sum_{j=m+1}^p \lambda_j \\ &\leq (1 + \lambda_{m+1} + \dots + \lambda_p) \epsilon, \end{aligned}$$

leading to $\|\mathbf{X}\mathbf{a}\| \leq \sqrt{(1 + \lambda_{m+1} + \dots + \lambda_p)\epsilon}$ as required. \square

Theorem 5.2 implies that if we are able to describe $\text{Span}(V)$, then we would be able to identify multicollinear relationships \mathbf{a} that exist in the design matrix \mathbf{X} , as Theorem 5.2(b) implies that every vector within $\sqrt{\epsilon}$ distance away from $\text{Span}(V)$ represents a $O(\sqrt{\epsilon})$ multicollinear relationship.

Identifying Multicollinear Relations

For $\dim(V) = r$, we have $r - 1$ linearly independent multicollinear relationships. There are infinite number of ways the basis of the $r - 1$ multicollinear relationships could be constructed, and different ways of constructing such bases lead to different constraints.

For example, assume that we have six variables $x_1, x, x_3, x_4, x_5, x_6$, and we know that $x_1 + x = x_3$ and $x_4 + x_5 = x_6$. Letting $\mathbf{a}_1 = (1, 1, -1, 0, 0, 0)^T$ and $\mathbf{a}_2 = (0, 0, 0, 1, 1, -1)^T$, we have $V = \text{Span}(\mathbf{a}_1, \mathbf{a}_2)$. Using Theorem 5.2 and ignoring \mathbf{b} as $\|\mathbf{b}\| = O(\sqrt{\epsilon})$, we can identify the two multicollinear relationships as \mathbf{a}_1 and \mathbf{a}_2 . Then, we add the constraints

$$z_1 + z + z_3 \leq 2, \quad z_4 + z_5 + z_6 \leq 2$$

However, there are alternative ways to characterize V in terms of two linearly independent vectors. Letting $\bar{\mathbf{a}}_1 = (1, 1, -1, 1, 1, -1)^T$ and $\bar{\mathbf{a}}_2 = (1, 1, -1, -1, -1, 1)^T$, then V is also $V = \text{Span}(\bar{\mathbf{a}}_1, \bar{\mathbf{a}}_2)$. Given this representation of V we would impose the constraints

$$z_1 + z + z_3 + z_4 + z_5 + z_6 \leq 4.$$

Note that the two sets of constraints are not equivalent.

It is important to identify the characterization of V that leads to the most stringent constraints to prevent multicollinearity. Towards this objective and ignoring the vector \mathbf{b} in Theorem 5.2, we introduce the idea of identifying a vector $\mathbf{a} \in \text{Span}(V)$ that has minimum support. We first compute the set $V = \{\mathbf{v}_1, \dots, \mathbf{v}_m\}$ of orthonormal eigenvectors with corresponding eigenvalues less than ϵ . We want to find a vector $\mathbf{a} \in \text{Span}(V)$ of minimum support. This is computed as follows:

$$\begin{aligned} \min & \quad \sum_{j=1}^m z_j \\ \text{subject to } & \quad \mathbf{a} = \sum_{i=1}^m \theta_i \mathbf{v}_i \\ & \quad |a_j| \leq M \cdot z_j, \quad j = 1, \dots, m \\ & \quad \left| \sum_{i=1}^m \theta_i \right| \geq \delta \\ & \quad z_j \in \{0, 1\}, \quad j = 1, \dots, m, \end{aligned} \tag{5.4}$$

where δ is a positive constant that ensures that $\mathbf{a} \neq 0$. Once the vector \mathbf{a} has been identified, we add the constraint

$$\sum_{i \in \text{Supp}(\mathbf{a})} z_i \leq |\text{Supp}(\mathbf{a})| - 1. \tag{5.5}$$

To continue the process of identifying new linearly independent multicollinear relationships, we add Eq. (5.5) to Problem (5.4), re-optimize the problem to identify a new multicollinear relationship, and identify a new constraint (5.5). We continue solving Problem (5.4) until the problem becomes infeasible, which means that we identified all linearly independent multicollinear relationships. Algorithm 5.1 determines all multicollinear relationships.

Algorithm 5.1 Iterative MIO for finding all linearly independent multicollinear relationships.

Input: E , the set of $\mathbf{X}^T \mathbf{X}$ eigenvectors with eigenvalues $< \epsilon$

Output: S , the characterization of V

```

1:  $S \leftarrow \emptyset$ 
2: while  $|S| < |E| - 1$  do
3:    $\mathbf{a}_0 \leftarrow$  solution of (5.4)
4:   if  $\mathbf{a}_0 \neq \emptyset$  then
5:      $S \leftarrow S \cup \mathbf{a}_0$ 
6:     Add the constraint  $\sum_{i \in Supp(\mathbf{a}_0)} z_i \leq |Supp(\mathbf{a}_0)| - 1$  to (5.4)
7:   else ▷ If (5.4) is Infeasible
8:     break
9: return  $S$ 

```

5.3 Examples of Holistic Regression

Given data \mathbf{X}, \mathbf{y} we summarize next our holistic linear regression framework:

$$\min_{\boldsymbol{\beta}, \mathbf{z}} \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \Gamma \|\boldsymbol{\beta}\|_1 \quad (5.6)$$

subject to:

$z_i, b_i \in \{0, 1\}$	$i = 1, \dots, p$
$-Mz_i \leq \beta_i \leq Mz_i,$	Big-M constraint ($i = 1, \dots, p$)
$\sum_{i=1}^p z_i \leq k$	Sparsity
$z_i = z_l$	Group Sparsity ($\forall i, j \in GS_m \forall m$)
$z_i + z_j \leq 1$	Pairwise Collinearity ($\forall i, j \in HC$)
$\sum_{i \in \mathcal{T}_j} z_i \leq 1, \quad j = 1, \dots, p$	Nonlinear transformations
$\frac{\beta_j}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} + Mb_j \geq t_\alpha z_j$	Significance ($j = 1, \dots, p$)
$-\frac{\beta_j}{\tilde{\sigma} \sqrt{(\mathbf{X}^T \mathbf{X})_{jj}^{-1}}} + M(1 - b_j) \geq t_\alpha z_j$	Significance ($j = 1, \dots, p$)
$\sum_{i \in Supp(\mathbf{a})} z_i \leq Supp(\mathbf{a}) - 1$	\forall multicollinear relations \mathbf{a}

identified by Algorithm 5.1.

Problem (5.6) is a mixed integer quadratic optimization (MIQO) model that can be solved in practical times by commercial solvers. To solve model (5.6) we need to specify the following parameters:

1. The robustness parameter Γ .
2. The sparsity parameter k .
3. The pairwise multicollinearity parameter ρ that affects the set HC .
4. The significance parameter α that affects the coefficient t_α .

As an example we illustrate the performance of holistic regression (5.6) on two datasets and compare it to a model that a modeler might develop using these data.

The Croq'Pain dataset

We compare model [5.6] with the approach that a human modeler might follow using the Croq'Pain dataset from [27]. The dataset originally comes from Croq'Pain, a French “restaurant rapide”, and contains data on sixty Croq'Pain stores. For each store, the dataset provides information on the store and the surrounding area. There are a total of sixteen variables provided per store (see Table 5.1 for details).

Table 5.1: Variables in the Croq'Pain dataset.

Variable	Description
EARN	Operating earnings in \$1000s
SIZE	Total area inside store
EMPL	Number of employees as of Dec. 31, 1994
P15	Number of 15-24 year olds in a 3 km radius
P25	Number of 25-34 year olds in a 3 km radius
P35	Number of 35-44 year olds in a 3 km radius
P45	Number of 45-54 year olds in a 3 km radius
P55	Number of people age 55+ in a 3 km radius
TOTAL	Total population in a 3 km radius
INC	Average income in town/neighborhood surrounding site
COMP	Number of competitors in 1 km radius
NCOMP	Number of restaurants that do not compete with Croq'Pain in a 1 km radius
NREST	Number of non-restaurant businesses in a 1 km radius
PRICE	Monthly rent per square meter of retail properties in the same locale
CLI	Cost of Living Index
K	Invested capital

The case described in [27] asks the student to use these data to build a regression model to help Croq’Pain decide whether to open a new store. The decision will be based on the store’s performance ratio, which is measured as the ratio of operating earnings to invested capital. The goal is to build a high quality regression model with performance ratio as the dependent variable, and the first step of this – the fitted model with all independent variables included – is given in [27]. Recall that we measure predictive quality using out of sample R^2 .

The Standard Approach

The model with all fourteen independent variables has an R^2 value of 0.867. Five of the fourteen variables are significant at the 0.05 level, and it seems that some of the coefficient estimates may take the opposite signs from what is expected. For example, the coefficients for number of employees and for total surrounding population are both negative.

A quick look at the correlation matrix shows that there are a number of independent variables which are highly correlated: for example, P35 and TOTAL have a correlation coefficient of 0.96. However, the 14×14 matrix is unwieldy to work with manually. Instead of trying to eliminate correlated variables first, we begin to refine this model by removing variables that are insignificant at the 0.05 level, starting with those with the lowest t -value. Removing variables one at a time according to this method until all variables left are significant results in a new model with only five independent variables (SIZE, P15, INC, NREST, and PRICE) and an training set R^2 value of 0.856. At this point, the number of independent variables is low enough to investigate the correlation matrix manually. None of the remaining five independent variables have correlation over 0.18 in magnitude, so we feel assured that multicollinearity is not a problem in this reduced model. We “sign-check” each of the remaining five independent variables and validate that the signs agree with our intuition. We move on to residual diagnostics, and check for normality of the residuals by plotting a histogram and for heteroscedasticity by plotting each of the independent variables against the residuals. There is no evidence of non-normality or of heteroscedasticity. Therefore, we use this model as the final model.

Holistic Regression

In the original case in [27], the students are first instructed to train their model using the entire dataset. The second part of the case asks them to rebuild using the first fifty data points to train the model and the last ten to validate. As holistic regression requires a training set and a validation set, we adopt the second option.

We run model (5.6) on the dataset using the following values for the parameters: $\rho = 0.8$, $\alpha = 0.05$ and $\Gamma = 1, \dots, 10$. We did not constrain k as the number of variables is already small. It takes less than 1 minute

to run, and returns a model with five independent variables: SIZE, P15, INC, NREST, and PRICE; exactly the same five we chose via the standard approach. The first four variables are significant at the 0.001 level, the last at the 0.01 level. The model has an out-of-sample R^2 value of 0.80.

With the Croq’Pain data, holistic regression and the standard approach produced the same model. In cases like these, we feel the main advantage of holistic regression is the amount of time saved from iterating through potential models. Although the computational time executing holistic regression is longer, the total time spent model-building is far shorter. In other cases, the standard approach may not lead to as clear of a path to a high-quality solution, or the dataset may contain enough variables to render it intractable for a human modeler. It is these cases for which holistic regression is not simply a time-saver, but a strong improvement over existing tools. The next example illustrates this case.

The Ames Housing Dataset

We consider the Ames Housing Dataset [80]. The dataset originally comes from the Ames City Assessor’s Office and contains data on property sales in Ames, Iowa between 2006 and 2010. The variables include discrete, continuous, nominal, and ordinal variables which describe the quality and quantity of physical attributes of each property sold. The physical attributes measured include building type and style, square footage and lot details, quality and materials of the property’s interior and exterior, and many more. The dataset was curated for use as a final group project in a semester-long regression class and is available along with full details in [80]. The prepared dataset contains 2930 observations and 80 variables. After expanding categorical variables into dummy variables and removing outliers and missing values, the final number of observations and variables is 2271 and 315, respectively. The objective is to use these data to build a regression model to predict housing sale prices.

The Standard Approach

This dataset is large and complex enough that there is no single clear best model. Indeed, this is the motivation in [80] behind assigning this dataset as a final course project; the richness of the data leads to fruitful discussion of students’ different approaches. [80] mentions that by using only the categorical variable for neighborhood and the two continuous variables that together comprise the property’s total square footage leads to a model that explains 80% of the variability. At the other end of the spectrum, the author also admits to spending a fair amount of time constructing a 36-variable model (using some variables he created through recoding and interactions) that explains 92% of the variation in sales. [80] does not give further details of the model. While this may be overly complicated, it illustrates the

challenge of building a high-quality regression model using the standard approach.

Holistic Regression

After removing missing values and splitting the dataset into training, validation, and test sets, we ran holistic regression on the dataset using the parameters $\rho = 0.8$, $\alpha = 0.05$, $k = 20$ and $\Gamma = 1, \dots, 10$. The best model generated contained twenty independent variables, all of which were significant at the 0.05 level, and had a test set R^2 value of 0.920. This is competitive with the predictive power of the more complicated 36-variable model constructed by [80], while being more interpretable and still retaining statistically significant variables. The best holistic regression model contained variables such as the overall quality and condition of the property, whether the property is identified as being in a particular neighborhood, the number of half bathrooms, whether the foundation of the home constructed from stone or not, the year the garage was built, various measurements of square footage, and the type of electrical system.

5.4 Insights from Computations

In this section, we illustrate the capabilities of holistic regression by demonstrating its performance on a wide variety of datasets. We include both synthetic and real-world datasets; and having both $n > p$ and $n < p$. Our goal is to demonstrate that all of the desirable characteristics outlined in Section 5.1 can be achieved with holistic regression in practical times.

We begin by examining basic datasets, where all variables are continuous and there is no special structure. In such datasets, the main properties we would like to ensure are interpretability (via general sparsity and limited pairwise multicollinearity constraints) and robustness (via a regularization parameter in the objective function). We consider synthetic examples to highlight holistic regression's performance on these properties individually and real datasets in which we look at desirable properties jointly. In each case, we compare holistic regression's performance to Lasso, as Lasso is designed to give interpretability and robustness.

We then provide results for datasets with additional features: datasets with the group sparsity property, with variables that need a nonlinear transformation, and so on. We again compare our results to Lasso, and compare it also to the published approach taken by the modeler, if available, or to specific algorithms designed for the setting at hand (e.g. group Lasso for the group sparsity case).

Synthetic Data

We generated data such that $\mathbf{x}_i \sim N(\mathbf{0}, \Sigma), i = 1, \dots, 2n$ are independent realizations from a p -dimensional multivariate normal distribution with mean zero and covariance matrix $\Sigma := (\sigma_{ij})$. The data was randomly split 50%/25%/25% into training, validation, and test set, respectively. The columns of the \mathbf{X} matrix were standardized such that the training set had columns with zero mean and unit ℓ_2 -norm. For a fixed $\mathbf{X}_{n \times p}$, we generated the response \mathbf{y} as follows: $\mathbf{y} = \mathbf{X} + \boldsymbol{\epsilon}$, where $\epsilon_i \stackrel{iid}{\sim} N(0, \sigma^2)$. We denote the number of nonzeros in $\boldsymbol{\beta}$ by k . The choice of $\mathbf{X}, \boldsymbol{\beta}, \sigma$ determines the Signal-to-Noise Ratio (SNR) of the problem, which is defined as:

$$\text{SNR} = \frac{\text{var}(\mathbf{x}' \boldsymbol{\beta})}{\sigma^2}.$$

In particular, we took $\sigma_{ij} = \rho^{|i-j|}$ for $i, j \in \{1, \dots, p\} \times \{1, \dots, p\}$. In our experiments, we consider $k = 10$ and $\beta_i = 1$ for $i \in \{1, \dots, p\}$ such that $i \bmod p/k = 0$ to generate k equally spaced values.

Real Data

We tested holistic regression on the following publicly-available datasets: Datasets White Wine Quality, Red Wine Quality, Yacht Hydrodynamics, and CPU obtained from the University of California Irvine Machine Learning Repository [7]. We obtained the datasets Elevator, Pyrimidines, and Compact from a data repository at the University of Porto [255]. We obtained the datasets LPGA 2008, LPGA 2009 and Airline Costs from a data repository at the University of Florida [272]. We obtained the Diabetes dataset from the `lars` package within R. The HIV dataset comes from the study [228] and is available at [129].

Computational Specifications

All computational tests were performed on a Dell Precision T7600 computer with an Intel Xeon E52687W (3.1 GHz) processor, 16 cores, and 128 GB of RAM. We used `Gurobi` [125] as the optimization solver, and implemented holistic regression in `Julia` [44]. We used `JuMP` [183]) to interface with Gurobi. We used the `GLMNet` package in Julia to compute Lasso solutions. We used the `grplasso` package in R ([223]) to compute group Lasso solutions.

Basic Structure

Our main goals are to achieve interpretability and robustness, while retaining predictive power. In order to judge how well holistic regression achieves interpretability we will report on the size k of the subset chosen, the maximum pairwise correlation, and the condition number of the final model.

To judge predictive power, we report the test set R^2 value. We compare these results to the size k of the subset chosen by Lasso, the maximum pairwise correlation in the Lasso model, and the test set R^2 value in the Lasso model. We selected the best regularization parameter for Lasso by its performance in the validation set. For the synthetic datasets, we also report the number of true positives achieved by each method.

We aim to return solutions in practical amounts of time, so we imposed a time limit on each optimization problem solved: 20 seconds in the $n > p$ case and 40 seconds in the $n < p$ case. Often optimality is reached before the time limit.

We present results in Tables 5.2 - 5.7 for synthetic datasets for the default parameters of holistic regression: ten values of Γ tested and 0.8 as the maximum pairwise correlation allowed. Each experiment corresponds to two rows in a table. The top row presents average results over five trials of the same experiment and the bottom row presents the standard error. We use the following notation:

1. SNR = signal-to-noise ratio,
2. k^* = value of k chosen by holistic regression,
3. TP = number of true nonzero variables identified by holistic regression,
4. Cor = the maximum pairwise correlation present in the final model, and
5. Cond = condition number.

Time for holistic regression is presented in hours, and is not meant to accurately benchmark the best possible time but to show that it is computationally tractable to solve these problems in a practical amount of time on standard computers.

Tables 5.2 and 5.3 show results for datasets designed to illustrate general sparsity, for the $n > p$ and $n < p$ case, respectively. Here we observe that holistic regression consistently identifies the true nonzero variables, and does not bring more than 1-2 additional noise variables into the model. In contrast, Lasso does correctly identify the true nonzero variables, but brings ≈ 24 noise variables into the model in the $n > p$ case and ≈ 45 noise variables into the model in the $n < p$ case. The holistic regression and Lasso models perform similarly in terms of predictive power (out of sample R^2).

Tables 5.4 and 5.5 show results for datasets designed to illustrate pairwise multicollinearity, for the $n > p$ and $n < p$ case, respectively. Again in these cases, the holistic regression and Lasso models perform similarly in terms of predictive power. However, the final Lasso models contain very high pairwise collinearity and condition numbers that indicate severe

multicollinearity issues. On the other hand, holistic regression returns models that generally have half or less of the maximum pairwise collinearity as the corresponding Lasso model, and the condition numbers do not show evidence of severe multicollinearity.

Tables 5.6 and 5.7 show results for datasets designed to illustrate robustness, for the $n > p$ and $n < p$ case, respectively. As described in Chapter 2, Lasso is designed to be robust to error in data. Indeed, in both the $n > p$ and $n < p$ case, holistic regression and Lasso achieve similar predictive power. The maximum pairwise collinearity and condition numbers of holistic regression are lower.

Table 5.2: Sparsity; $n = 500$, $p = 100$, $\rho = 0$, $\Delta \mathbf{X} = \mathbf{0}$.

SNR	HR k*	TP	R ²	Cor	Cond	Time
6.32	10.6	10	0.716	0.119	1.61	0.448
	0.358	0	0.007	0.007	0.02	0.011
3.16	10.6	10	0.909	0.119	1.27	0.439
	0.358	0	0.003	0.007	0.29	0.011
1.58	10	10	0.975	0.117	1.58	0.304
	0	0	0.001	0.007	0.04	0.011
SNR	Lasso k*	TP	R ²	Cor	Cond	
6.32	34.8	10	0.701	0.148	2.782	
	3.51	0	0.007	0.010	0.127	
3.16	34.4	10	0.904	0.148	2.805	
	3.72	0	0.002	0.010	0.146	
1.58	34.6	10	0.974	0.160	2.797	
	4.40	0	0.001	0.016	0.194	

We present results on real-world data in Table 5.8. All optimization problems were solved to optimality except for the diabetes and HIV datasets where a time limit of 20 seconds per optimization problem solved was enforced. Note that n here indicates the size of the training dataset – the original dataset has $2n$ observations.

Holistic regression achieves similar predictive performance to Lasso, but is significantly more interpretable, choosing fewer variables in general and successfully limiting the degree of multicollinearity present in the final model.

We notice that the Pyrimidines dataset has significantly lower predictive power than Lasso. This is the price of insisting on interpretability, despite a relatively low ratio of observations to variables. We demonstrate the holistic regression’s performance on this dataset when the maximum correlation threshold is set to 1 (i.e., no limit) and record the performance

Table 5.3: Sparsity; $n = 100$, $p = 500$, $\rho = 0$, $\Delta \mathbf{X} = \mathbf{0}$.

SNR	HR k*	TP	R²	Cor	Cond	Time
10.54	10.2 0.179	10 0	0.991 0.000	0.249 0.026	2.664 0.195	1.00 0.08
6.32	10 0	10 0	0.976 0.001	0.231 0.018	2.793 0.217	1.18 0.00
3.16	11 0.283	10 0	0.896 0.006	0.216 0.012	3.348 0.192	1.64 0.42
SNR	Lasso k*	TP	R²	Cor	Cond	
10.54	55 7.33	10 0	0.982 0.001	0.323 0.006	139 97	
6.32	56 8.78	10 0	0.952 0.003	0.323 0.006	1472 1290	
3.16	58.2 9.10	10 0	0.813 0.012	0.343 0.014	5215 4634	

Table 5.4: Pairwise Multicollinearity; $n = 500$, $p = 100$, True $k = 10$, $\rho = 0.9$, $\Delta \mathbf{X} = \mathbf{0}$.

SNR	HR k*	TP	R²	Cor	Cond	Time
8.73	10.00 0.00	10.00 0.00	0.99 0.00	0.40 0.01	4.15 0.17	0.30 0.02
4.37	10.40 0.36	10.00 0.00	0.95 0.00	0.47 0.07	5.65 1.25	0.34 0.04
2.18	11.40 0.54	9.60 0.22	0.81 0.01	0.63 0.08	7.92 2.25	0.63 0.15
SNR	Lasso k*	TP	R²	Cor	Cond	
8.73	34.40 2.65	10.00 0.00	0.99 0.00	0.91 0.00	126.28 13.15	
4.37	37.20 3.66	10.00 0.00	0.94 0.00	0.91 0.00	146.36 20.83	
2.18	36.60 3.37	10.00 0.00	0.81 0.01	0.91 0.00	142.17 18.89	

in Table 5.9.

In these cases it is up to the analyst to judge which model is preferable; one with better predictive performance or one with coefficients that are more interpretable. The benefit of using holistic regression is that it is

Table 5.5: Pairwise Multicollinearity; $n = 100$, $p = 500$, True $k = 10$, $\rho = 0.8$, $\Delta \mathbf{X} = \mathbf{0}$.

SNR	HR k*	TP	R ²	Cor	Cond	Time
10.54	10.4	10	0.990	0.331	5.58	1.99
0	0.358	0	0.001	0.089	1.48	0.517
SNR	Lasso k*	TP	R ²	Cor	Cond	
6.32	10.4	10	0.976	0.436	6.11	2.12
0	0.219	0	0.003	0.118	1.24	0.395
3.16	12.4	8.8	0.835	0.433	4.38	2.11
0	0.219	0.72	0.041	0.099	0.51	0.514
	56.2	10	0.979	0.850	119.8	
	2.92	0	0.004	0.005	8.2	
	57	10	0.941	0.846	122.19	
	2.65	0	0.010	0.005	7.47	
	61.2	9.8	0.768	0.846	245.4	
	3.70	0.179	0.029	0.005	117.9	

Table 5.6: Robustness: $n = 500$, $p = 100$, True $k = 10$, $\rho = 0$, $\Delta \mathbf{X} \sim \text{Uniform}(0,2)$.

SNR	HR k*	TP	R ²	Cor	Cond	Time
6.32	10	10	0.975	0.117	1.58	0.448
	0.000	0	0.001	0.007	0.04	0.011
3.16	10.6	10	0.909	0.119	1.27	0.439
	0.358	0	0.003	0.007	0.29	0.011
1.58	10.6	10	0.716	0.119	1.61	0.304
	0.358	0	0.007	0.007	0.02	0.011
SNR	Lasso k*	TP	R ²	Cor	Cond	
6.32	34.6	10	0.974	0.160	2.797	
	4.40	0	0.001	0.016	0.194	
3.16	34.4	10	0.904	0.148	2.805	
	3.72	0	0.002	0.010	0.146	
1.58	34.8	10	0.701	0.148	2.782	
	3.51	0	0.007	0.010	0.127	

simple for an analyst to tweak the parameters and quickly understand the tradeoffs.

Table 5.7: Robustness: $n = 100$, $p = 500$, True $k = 10$, $\rho = 0$, $\Delta \mathbf{X} \sim \text{Uniform}(0,1)$.

SNR	HR k*	TP	R ²	Cor	Cond	Time
10.54	10.6	9.6	0.880	0.282	2.777	1.173
0	0.607	0.358	0.034	0.021	0.131	0.000
6.32	10	9.4	0.828	0.246	2.716	1.643
0	0.632	0.358	0.033	0.017	0.268	0.419
3.16	11	9.4	0.769	0.262	2.475	1.876
0	0.85	0.358	0.030	0.027	0.585	0.420
SNR	Lasso k*	TP	R ²	Cor	Cond	
	53.8	10	0.856	0.376	53.8	
	4.66	0	0.013	0.018	22.5	
	53.4	10	0.829	0.357	107.3	
	7.69	0	0.029	0.014	75.3	
	61.8	10	0.705	0.338	763.0	
	10.27	0	0.035	0.010	546.8	

Table 5.8: Results for Basic Structure Real Datasets.

Dataset	n	p	HR k*	R ²	Cor	Lasso k*	R ²	Cor
CPU	105	6	5	0.869	0.716	6	0.861	0.716
Yacht	154	6	1	0.600	NA*	1	0.602	NA*
White Quality	2499	11	10	0.270	0.619	9	0.280	0.828
Red Quality	800	11	6	0.384	0.40	7	0.386	0.69
Compact	4096	21	15	0.717	0.733	21	0.725	0.942
Elevator	8280	18	10	0.808	0.678	15	0.809	0.999
Pyrimidines	37	26	15	0.175	0.781	20	0.367	0.928
LPGA 2008	78	6	2	0.877	0.02	3	0.873	0.234
LPGA 2009	73	11	7	0.814	0.784	10	0.807	0.943
Airline Costs	15	9	2	0.672	0.501	9	0.390	0.973
Diabetes	221	64	4	0.334	0.423	14	0.381	0.672
HIV	528	98	11	0.945	0.662	39	0.944	0.760

*Note that both holistic regression and Lasso choose only one independent variable for the Yacht Hydrodynamics dataset, hence there is no maximum pairwise correlation in this case.

Special Structure

Nonlinear Transformations

We investigate holistic regression's capability to identify when a nonlinear transformation of an independent variable may be useful. For this task, we

Table 5.9

Dataset	n	p	HR k*	R ²	Cor	Lasso k*	R ²	Cor
Pyrimidines	37	26	18	0.375	0.870	20	0.367	0.928

used the Concrete Compressive Strength dataset from [276] available in the UCI Machine Learning Repository [7]. The dataset contains 8 independent variables and 1030 observations. As before, we randomly split the dataset into a training set (50%), validation set (25%), and test set (25%).

Table 5.10: Independent Variables in the Concrete Compressive Strength Dataset.

Variable	Units
Concrete Compressive Strength	MPa
Cement	kg/m ³
Blast Furnace Slag	kg/m ³
Fly Ash	kg/m ³
Water	kg/m ³
Superplasticizer	kg/m ³
Coarse Aggregate	kg/m ³
Fine Aggregate	kg/m ³
Age	day

The independent variable is the compressive strength of concrete, and the dependent variables are the ingredients as well as the age of the concrete (see Table 5.10 for details). The practical goal in civil engineering is to design a concrete mixture which will have high compressive strength. However, concrete compressive strength is known to be a highly nonlinear function of its age and ingredients.

On the original data, both holistic regression and Lasso chose to use all covariates and produced a test set R^2 of 0.609. We then reran holistic regression with an extended dataset, which contained each of the original columns x as well as three transformed versions of each column: x^2 , \sqrt{x} , and $\log(x)$. For the variables which take zero values (blast furnace slag, fly ash, and superplasticizer), we adjusted the log transformation to be $\log(x + 0.00001)$. We added the constraint that at most one of x , x^2 , \sqrt{x} , and $\log(x)$ appeared in the final model.

As expected, the inclusion of transformed covariates significantly improved upon the models created with just the original variables. Holistic regression selected six covariates to appear in the top model. The six covariates chosen were blast furnace slag, water, $\log(\text{fly ash})$, $\log(\text{super plasticizer})$, $\log(\text{day})$, and $\text{cement}^{1/2}$. Each covariate was significant at the

Table 5.11: Independent Variables in the Energy Efficiency Dataset.

Variable	Type
Relative Compactness	Continuous
Surface Area	Continuous
Wall Area	Continuous
Roof Area	Continuous
Overall Height	Continuous
Orientation	Categorical; 4 levels
Glazing Area	Continuous
Glazing Area Distribution	Categorical; 6 levels

$\alpha = 0.001$ level and test set R^2 was 0.823, a significant improvement over the original test set R^2 of 0.609.

We also tested Lasso on the dataset that included the nonlinear transformations. Lasso selected a model with twelve covariates which resulted in a test set R^2 of 0.834. In addition to the first five covariates chosen by holistic regression, Lasso also selected cement², super plasticizer², day², log(cement), log(coarse aggregate), log(fine aggregate), and $\sqrt{\text{cement}}$. In our opinion, the minor increase in test set R^2 does not warrant using a significantly less interpretable model.

Although the number of variables went from 8 to 32 when we included nonlinear transformations, holistic regression took the same amount of time (roughly 1-1.5 minutes) to execute in both cases. By imposing limiting constraints on transformations and pairwise correlation, the feasible space is not significantly enlarged by including nonlinear transformations.

Group Sparsity

We demonstrate our results in a group sparsity setting using the Energy Efficiency dataset from [260] available on the UCI Machine Learning Repository ([7]). The dataset has 768 observations of six continuous independent variables and two categorical independent variables. The independent variables describe building properties (see Table 5.11 for details). There are two dependent variables available: heating load and cooling load. We test our method on both dependent variables.

The binary expansion of the categorical variable orientation into three new binary variables and of glazing area distribution into five new binary variables meant that the dataset passed to holistic regression contained fourteen independent variables. When we ran holistic regression, the best model contained three variables: wall area, overall height, and glazing area. We found identical results when predicting cooling load. holistic regression chose not to use either of the categorical variables provided. The top heating

load model had test set R^2 of ≈ 0.88 and the top cooling load model had test set R^2 values of ≈ 0.85 .

In [260], the original study of this dataset, the authors found that wall area, roof area, and relative compactness were the variables that appear mostly associated with heating load and cooling load, although all variables appear in their model. Using the Random Forest method, they also found importance scores for each variable and found that glazing area was the most important variable. They note that “Interestingly, the most important variable (glazing area) is not the most correlated with either output variable. From an engineering perspective, it can be intuitively understood that the glazing area is of paramount significance...”

We find it notable that holistic regression did not identify the same three variables as most critical for predicting the responses, and could not have: due to correlation of -0.86 between roof area and relative compactness, these two variables could not have both been in holistic regression’s final model. However, glazing area, which the authors point out as having paramount significance, is in all three of our top models for both response variables.

We also tested group lasso. The group Lasso models for predicting the two response variables each chose to use thirteen of the fourteen variables, including both categorical variables. The only variable excluded was surface area. The heating load model had a test set R^2 of ≈ 0.91 and the cooling load model had a test set R^2 value of ≈ 0.86 .

Significance

In this section, we demonstrate that holistic regression leads to statistical significance in an efficient way using both synthetic and real-world data sets.

Synthetic Data. We aim to recover the true β with varying n, p , support k and error types. The β_j is uniformly chosen between $[-10, 10]$ and $X_{ij} \sim N(0, 1)$ for each i, j in the matrix independently. We monitor three statistics of the recovered $\tilde{\beta}$:

- **Accuracy** - the percentage of non-zero β ’s in the ground truth we identify correctly:

$$ACC = \frac{Supp(\beta) \cap Supp(\tilde{\beta})}{Supp(\beta)}.$$

- **False Positive Rate** - the percentage of non-zero β ’s recovered that are zero in the ground truth. It is defined as:

$$FPR = \frac{Supp(\tilde{\beta}) \setminus Supp(\beta)}{Supp(\tilde{\beta})}.$$

Table 5.12: Holistic regression performance for statistical significance using synthetic data for $\alpha = 0.05$.

n	Error	p	k	ACC	FPR	Time
100	$N(0, 0.1)$	10	3	100%	7.4%	0.65s
200	$N(0, 0.1)$	10	3	100%	4.5%	2.9s
500	$N(0, 0.1)$	10	3	100%	0%	6.67 s
1000	$N(0, 0.1)$	10	3	100%	0%	36.4s
500	$N(0, 1)$	10	3	100%	26%	7.93s
1000	$N(0, 1)$	10	3	100%	0%	38.9s
500	t_5	10	3	95%	35%	6.75s
1000	t_5	10	3	100%	21%	35.2s

- **Time** - the time the algorithm takes to converge to a final solution.

In addition to solving (5.6) We chose $\alpha = 0.05$ as the significance level and present the results in Table 5.12.

Real-World Data. We present the performance of the t -test constraints using five real-world datasets. We randomly split the dataset into 60%/20%/20% as training, validation, and testing, and average the results 10 times. We record the support k^* of the β in the final model, and its mean squared loss of its predictions on the testing set, along with the time it took for the model to converge. We chose $\alpha = 0.05$ as the significance level and present the results in Table 5.13.

Table 5.13: Holistic regression performance relative for statistical significance using real-world data for $\alpha = 0.05$.

Dataset	p	k^*	Loss	Time
NCAA	11	6	39.9	17.3s
Pollution	6	4	249.8	2.74s
Diabetes	6	2	1553.2	9.4s
Baseball	6	4	6397.3	27.7s
Pyrimidine	11	7	0.5970	2.3s

Multicollinearity detection

In this section, we use synthetic data to evaluate the performance of Algorithm 5.1.

We model the design matrix \mathbf{X} such that $X_{ij} \sim N(0, 1)$ independently for each $i \in \{1, \dots, n\}$, $j \in \{1, \dots, p\}$. Then, we randomly select certain number of columns to be replaced by linear combinations of other columns $\sum_{j \in S} \gamma_j \mathbf{X}_j$. The parameters γ_i are selected randomly from the uniform distribution $[-10, 10]$, and we control S as follows:

1. We first determine the number q of variables we want to involve in this multicollinear relationship.
2. We randomly select q numbers from $\{1, \dots, p\}$ without replacement, and denote that set S .

We add noise $\tilde{\mathbf{X}}$ according to the distribution indicated in Table 5.14, and evaluate the performance of Algorithm 5.1 on $\mathbf{X} + \tilde{\mathbf{X}}$.

Algorithm 5.1 performance is evaluated on accuracy and false positive rate of the multicollinear relationships found, along with the time taken for the algorithm to converge.

The columns in Table 5.14 labeled 3, 4 indicate the number of multicollinear relationships involving 3, 4 variables that have been introduced into the data. For 4+, we randomly selected a number within $\{5, 6, 7, 8, 9, 10\}$ to be the number of variables involved in the multicollinear relationship. Every experiment below was repeated 10 times then averaged.

Table 5.14: Performance of Algorithm 5.1 for multicollinearity detection.

n	p	3	4	4+	Noise	ACC	FPR	Time
1000	100	3	1	1	$N(0, 0.01)$	100%	0%	0.27s
1000	500	3	1	1	$N(0, 0.01)$	100%	0%	2.37s
1000	1000	3	1	1	$N(0, 0.01)$	100%	5%	520.23s
1000	500	5	3	2	$N(0, 0.01)$	100%	0%	33.40s
1000	1000	5	3	2	$N(0, 0.01)$	100%	24%	5940.56s
1000	500	3	1	1	$N(0, 0.03)$	100%	0%	2.29s
1000	1000	3	1	1	$N(0, 0.03)$	100%	11%	32.17s

Table 5.14 shows that Algorithm 5.1 is well suited for n, p up to the thousands and could detect multicollinearity with high accuracy and low false positive rates.

5.5 Concluding Remarks

In this chapter, we have leveraged the power of MIQO and proposed holistic regression, an approach for incorporating a variety of desired properties into a linear regression model. Holistic regression provides the only methodology we are aware of to construct models that impose statistical properties simultaneously. This results in a generally applicable, unified framework for addressing all aspects of the model-building process. Using both real and synthetic data, we demonstrate that holistic regression produces high-quality linear regression models in realistic timelines.

5.6 Notes and Sources

The material of this chapter is from [30]. The approach to model significance and global multicollinearity is from [32]. [31] have generalized the holistic regression framework to logistic regression.

Chapter 6

Sparse Classification

All models are wrong, but some are useful.

– George Box

Contents

- 6.1. Regularized classification
- 6.2. The dual method to sparse classification
- 6.3. Sparse logistic regression
- 6.4. Sparse Support Vector Machine
- 6.5. Concluding Remarks
- 6.6. Notes and Sources

Sparse classification is a central problem in machine learning as it leads to more interpretable models. Given data $\{(\mathbf{x}_i, y_i)\}_{i=1,\dots,n}$ with $y_i \in \{-1, 1\}$ and $\mathbf{x}_i \in \mathbb{R}^p$, we aim at computing the vector $\boldsymbol{\beta}$ which minimizes an empirical loss ℓ subject to the constraint that the number of its nonzero entries does not exceed some value k :

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \sum_{i=1}^n \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) \text{ s.t. } \|\boldsymbol{\beta}\|_0 \leq k. \quad (6.1)$$

Despite its conceptual appeal, Problem (6.1) is an NP-hard optimization problem [202]. Thus, much of the literature has focused on heuristic proxies and replaced the 0-norm with convex norms which are known to lead to sparser models [6]. Even though regularization enforces robustness rather than sparsity as we saw in Chapter 2, the ℓ_1 -penalty formulation

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \sum_{i=1}^n \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \lambda \|\boldsymbol{\beta}\|_1, \quad (6.2)$$

is abundantly used in practice. Efficient numerical algorithms exist [105], off-the-shelf implementations are publicly available [106] and recovery of the true sparsity is theoretically guaranteed under some assumptions on the data.

In this chapter, we adapt and generalize the dual algorithm from Chapter 3 to formulate exact sparse classification as a binary convex optimization problem and propose a tractable numerical algorithm to solve it in high dimensions. In Chapter 3, we addressed linear regression for which a closed-form solution exists and made extensive use of this closed-form solution. The framework in this chapter extends to cases where a closed-form solution is not available and includes logistic regression and SVM with one or two norm. We demonstrate the tractability of the algorithm in practice for two-class logistic regression and Support Vector Machines on data sets for which n and p are in the 10,000s. Finally we also observe phase transition phenomena we also observed in Chapter 3.

6.1 Regularized classification

Two very popular classification methods in Machine Learning are logistic regression and Support Vector Machine (SVM). Despite different motivations and underlying intuition, both methods lead to a similar formulation which can be addressed under the unifying lens of regularized classification:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^p} \sum_{i=1}^n \ell(y_i, \boldsymbol{\beta}^T \mathbf{x}_i) + \frac{1}{2\gamma} \|\boldsymbol{\beta}\|_2^2, \quad (6.3)$$

where ℓ is an appropriate loss function and γ a regularization coefficient.

In the logistic regression framework, the loss function is the logistic loss

$$\ell(y, u) = \log(1 + e^{-yu}),$$

and the objective function can be interpreted as the negative log-likelihood of the data plus a regularization term, which ensures strict convexity of the objective and existence of an optimal solution.

In the SVM framework, the loss function ℓ is the hinge loss:

$$\ell(y, u) = \max(0, 1 - yu).$$

Under proper normalization assumptions, the square norm $\|\beta\|_2^2$ relates to the notion of margin, which characterizes the robustness of the separating hyperplane $\{\mathbf{x} : \beta^T \mathbf{x} = 0\}$, while the loss part penalizes the data points which do not satisfy $y_i \beta^T \mathbf{x} \geq 1$, that is points which are misclassified or lie within the margin [264].

In addition, this general formulation (6.3) can be extended to a broad family of other loss functions used in classification (e.g. 2-norm SVM) or even in regression problems. Throughout the chapter we make the following assumption:

Assumption 6.1. *The loss function $\ell(y, \cdot)$ is convex for $y \in \{-1, 1\}$.*

As we have seen in Chapter 3, deeper results and insights can typically be obtained by adopting a dual perspective. Denoting $\mathbf{X} = (\mathbf{x}_i^T)_{i=1,\dots,n} \in \mathbb{R}^{n \times p}$ the design matrix, we have:

Theorem 6.1. *Under Assumption 6.1, strong duality holds for problem (6.3) and its dual is*

$$\max_{\alpha \in \mathbb{R}^n} -\sum_{i=1}^n \hat{\ell}(y_i, \alpha_i) - \frac{\gamma}{2} \alpha^T \mathbf{X} \mathbf{X}^T \alpha \quad \text{s.t. } \mathbf{e}^T \alpha = 0, \quad (6.4)$$

where $\hat{\ell}(y, \alpha) := \max_{u \in \mathbb{R}} u\alpha - \ell(y, u)$ is the Fenchel conjugate of the loss function ℓ .

Table 6.1 summarizes some popular loss functions in classification and their corresponding Fenchel conjugates.

Proof. For regularized classification, we have that

$$\min_{\beta} \sum_{i=1}^n \ell(y_i, \beta^T \mathbf{x}_i) + \frac{1}{2\gamma} \|\beta\|_2^2 = \min_{\beta, z} \sum_{i=1}^n \ell(y_i, z_i) + \frac{1}{2\gamma} \|\beta\|_2^2 \quad \text{s.t. } z_i = \beta^T \mathbf{x}_i.$$

By Assumption 6.1, the objective is convex, the optimization set is convex and Slater's conditions hold. Hence, strong duality must hold and the

Table 6.1: Examples of loss functions and their corresponding Fenchel conjugates, as defined in Theorem 6.1.

Method	Loss $\ell(\mathbf{y}, \mathbf{u})$	Fenchel conjugate $\hat{\ell}(\mathbf{y}, \alpha)$
Logistic loss	$\log(1 + e^{-yu})$	$(1 + y\alpha)\log(1 + y\alpha) - y\alpha \log(-y\alpha)$, if $y\alpha \in [-1, 0]$, and $+\infty$, otherwise.
1-norm SVM	$\max(0, 1 - yu)$	$\begin{cases} y\alpha, & \text{if } y\alpha \in [-1, 0], \\ +\infty, & \text{otherwise.} \end{cases}$
2-norm SVM	$1/2 \max(0, 1 - yu)^2$	$\begin{cases} 1/2\alpha^2 + y\alpha, & \text{if } y\alpha \leq 0, \\ +\infty, & \text{otherwise.} \end{cases}$

primal is equivalent to the dual problem. To derive the dual formulation, we introduce Lagrange multipliers α_i associated with the equality constraints:

$$\begin{aligned}
& \min_{\beta, \mathbf{z}} \sum_{i=1}^n \ell(y_i, z_i) + \frac{1}{2\gamma} \|\beta\|_2^2 \text{ s.t. } z_i = \beta^T \mathbf{x}_i \\
&= \min_{\beta, \mathbf{z}} \sum_{i=1}^n \ell(y_i, z_i) + \frac{1}{2\gamma} \|\beta\|_2^2 + \max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^n \alpha_i (\beta^T \mathbf{x}_i - z_i) \\
&= \min_{\beta, \mathbf{z}} \max_{\alpha} \left(\sum_{i=1}^n \ell(y_i, z_i) - \alpha_i z_i \right) + \left(\frac{1}{2\gamma} \|\beta\|^2 + \beta^T \left[\sum_i \alpha_i \mathbf{x}_i \right] \right) \mathbf{e}^T \boldsymbol{\alpha} \\
&= \max_{\boldsymbol{\alpha}} \sum_{i=1}^n \min_{z_i} (\ell(y_i, z_i) - \alpha_i z_i) + \min_{\beta} \left(\frac{1}{2\gamma} \|\beta\|^2 + \beta^T \mathbf{X}^T \boldsymbol{\alpha} \right).
\end{aligned}$$

Let us consider the inner minimization problems separately. First,

$$\min_{z_i} (\ell(y_i, z_i) - \alpha_i z_i) = -\max_{z_i} (\alpha_i z_i - \ell(y_i, z_i)) = -\hat{\ell}(y_i, \alpha_i).$$

Then, $1/2\gamma \|\beta\|^2 + \beta^T \mathbf{X}^T \boldsymbol{\alpha}$ is minimized at β^* satisfying: $1/\gamma \beta^* + \mathbf{X}^T \boldsymbol{\alpha} = 0$. Hence

$$\min_{\beta} \left(\frac{1}{2\gamma} \|\beta\|^2 + \beta^T \mathbf{X}^T \boldsymbol{\alpha} \right) = -\frac{1}{2\gamma} \|\beta^*\|^2 = -\frac{\gamma}{2} \boldsymbol{\alpha}^T \mathbf{X} \mathbf{X}^T \boldsymbol{\alpha},$$

leading to (6.4). \square

The derivation of the dual (6.4) reveals that the optimal primal variables β^* and dual variables $\boldsymbol{\alpha}^*$ satisfy

$$\beta^* = -\gamma \mathbf{X}^T \boldsymbol{\alpha}^*.$$

In other words, β^* is a linear combination of some data points of X . Such an observation has historically led to the intuition that β^* was *supported* by some observed vectors \mathbf{x}_i and the name Support Vector Machine was coined [75].

Moreover, the dual point of view opens the door to non-linear classification using kernels [240]. The positive semi-definite matrix \mathbf{XX}^T , often referred to as the kernel or Gram matrix, is central in the dual problem (6.4) and could be replaced by any kernel matrix \mathbf{K} whose entries K_{ij} encode some measure of similarity between inputs \mathbf{x}_i and \mathbf{x}_j .

Numerical algorithms There is a rich literature on numerical algorithms for solving either the primal (6.3) or the dual (6.4) formulation for the regularized classification problem in the case of logistic regression and SVM. Gradient descent or Newton-Raphson methods are well-suited when the loss function is smooth. In addition, in the case where the dual problem is constrained to $\mathbf{e}^T \alpha = 0$, particular step size rules or trust regions [174] can be implemented to cope with such linear constraints.

6.2 The dual method to sparse classification

As discussed in Chapter 3, sparsity is a highly desirable property for statistical estimators, especially in high-dimensional regimes ($p > n$) such as the ones encountered in biological applications, where interpretability is crucial. A natural way to induce sparsity is to add a constraint on the number of nonzero coefficients of β and solve:

$$\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^n \ell(y_i, \beta^T \mathbf{x}_i) + \frac{1}{2\gamma} \|\beta\|_2^2 \quad \text{s.t. } \|\beta\|_0 \leq k. \quad (6.5)$$

In the next theorem we express (6.5) as a convex binary optimization problem:

Theorem 6.2. *Problem (6.5) is equivalent to*

$$\min_{\mathbf{s} \in S_k^p} c(\mathbf{s}), \quad (6.6)$$

where for any $\mathbf{s} \in \{0, 1\}^p$,

$$c(\mathbf{s}) := \max_{\alpha \in \mathbb{R}^n} f(\alpha, \mathbf{s}) = - \sum_{i=1}^n \hat{\ell}(y_i, \alpha_i) - \frac{\gamma}{2} \sum_{j=1}^n s_j \alpha^T \mathbf{X}_j \mathbf{X}_j^T \alpha \quad \text{s.t. } \mathbf{e}^T \alpha = 0. \quad (6.7)$$

In particular, $c(\mathbf{s})$ is convex over $[0, 1]^p$.

The proof is similar to Theorem 3.19 and is omitted. In practice, for a given support \mathbf{s} , we can evaluate the function $c(\mathbf{s})$ by solving the

maximization problem (6.7) with any of the numerical procedures presented in the previous section. In what follows we need to calculate the gradient of the function c as well. Using the dual maximizer $\alpha^*(\mathbf{s})$ in (6.7) at a support \mathbf{s} , we can at no additional computational cost obtain the gradient of c too. Indeed, it follows that

$$\frac{\partial c(\mathbf{s})}{\partial s_j} = -\frac{\gamma}{2} \boldsymbol{\alpha}^*(\mathbf{s})^T \mathbf{X}_j \mathbf{X}_j^T \boldsymbol{\alpha}^*(\mathbf{s}).$$

A cutting-plane procedure

We aim to solve the convex binary optimizaton problem (6.6), taking into account that we can readily compute $c(\mathbf{s})$ and $\nabla c(\mathbf{s})$ for any given \mathbf{s} . None of the commercial solvers available are targeted to solve such CIO problems where there is no closed-form expression for $c(\mathbf{s})$. We first reformulate (6.6) as a mixed-integer optimization problem in epigraph form

$$\min_{\mathbf{s} \in S_k^p, \eta} \eta \text{ s.t. } \eta \geq c(\mathbf{s}). \quad (6.8)$$

We find a solution to (6.6) by iteratively constructing a piece-wise linear lower approximation of c . The solver structure is given in pseudocode in Algorithm 6.1.

Algorithm 6.1 Outer-approximation algorithm

Input: : $\mathbf{X} \in \mathbb{R}^{n \times p}$, $\mathbf{y} \in \{-1, 1\}^p$, $k \in \{1, \dots, p\}$, \mathbf{s}_1 .
Output: An optimal solution \mathbf{s}^* to (6.6).

```

 $\mathbf{s}_1 \leftarrow$  warm-start
 $\eta_1 \leftarrow 0$ 
 $t \leftarrow 1$ 
repeat
     $\mathbf{s}_{t+1}, \eta_{t+1} \leftarrow \text{argmin}_{\mathbf{s}, \eta} \{ \eta : \mathbf{s} \in S_k^p, \eta \geq c(\mathbf{s}_i) + \nabla c(\mathbf{s}_i)^T (\mathbf{s} - \mathbf{s}_i), i \in [t] \}$ 
     $t \leftarrow t + 1$ 
until  $\eta_t < c(\mathbf{s}_t)$  return  $\mathbf{s}_t$ 
```

Theorem 6.3. [99] Under Assumption 6.1, Algorithm 6.1 terminates in a finite number of steps and returns an optimal solution of (6.6).

We next provide numerical evidence that Algorithm 6.1 is indeed extremely efficient in practice, both for logistic and hinge loss functions.

Practical implementation considerations

We implemented Algorithm 6.1 using lazy constraints, a feature that integrates the cutting-plane procedure within a unique branch-and-bound enumeration tree, shared by all subproblems.

As for the ElasticNet, sparse regularized classification (6.5) involves two hyperparameters - a regularization factor γ and the sparsity k . For the regularization parameter γ , we fit its value using cross-validation among values uniformly distributed in the log-space: we start with a low value γ_0 (typically γ_0 scaling as $1/\max_i \|\mathbf{x}_i\|^2$).

Finally, Algorithm 6.1 requires an initialization value \mathbf{s}_1 . Concerned both by statistical relevance and ease of implementation, we use the Lasso estimator provided by the GLMNet package [106].

6.3 Sparse logistic regression

To demonstrate the practability of the Algorithm 6.1, we consider the logistic loss function

$$\ell(y, u) = \log(1 + e^{-yu}).$$

Its Fenchel conjugate is

$$\begin{aligned}\hat{\ell}(y, \alpha) &= \max_{u \in \mathbb{R}} u\alpha - \log(1 + e^{-yu}) \\ &= \begin{cases} (1 + y\alpha) \log(1 + y\alpha) - y\alpha \log(-y\alpha), & \text{if } y\alpha \in [-1, 0], \\ +\infty, & \text{otherwise,} \end{cases}\end{aligned}$$

so we can apply Algorithm 6.1, the conditions $y_i\alpha_i \in [-1, 0]$ being implemented as additional linear constraints on dual variables α , to synthetic and real-world data sets.

Simulations on synthetic data

We constructed data sets on which we assess Algorithm 6.1 and compare it to a state-of-the-art Lasso procedure. We draw $\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}_p, \Sigma)$, $i = 1, \dots, n$ independent realizations from a p -dimensional normal distribution with mean $\mathbf{0}_p$ and covariance matrix $\Sigma_{ij} = \rho^{|i-j|}$. Columns of \mathbf{X} are then normalized to have zero mean and unit variance. We randomly sample a weight vector $\beta_{true} \in \{-1, 0, 1\}^p$ with exactly k nonzero coefficients. We draw ε_i , $i = 1, \dots, n$, i.i.d. noise components from a normal distribution scaled according to a chosen signal-to-noise ratio

$$\sqrt{SNR} = \|\mathbf{X}\beta_{true}\|_2 / \|\varepsilon\|_2.$$

Finally, we construct y_i according to a logistic model

$$\mathbb{P}(Y = y | \mathbf{X} = \mathbf{x}_i) = \frac{1}{1 + e^{-y(\beta_{true}^T \mathbf{x}_i + \varepsilon_i)}}.$$

Such a methodology enables us to produce synthetic data sets where we control the sample size n , feature size p , sparsity k , feature correlation ρ and signal-to-noise ratio SNR .

Support recovery metrics

Given the true classifier β_{true} of sparsity k_{true} , we assess the correctness of a classifier β of sparsity k by its accuracy, i.e., the number of true features it selects

$$A(\beta) = |\{j : w_j \neq 0, \beta_{true,j} \neq 0\}| \in \{0, \dots, k_{true}\},$$

and the false discovery, i.e., the number of false features it incorporates

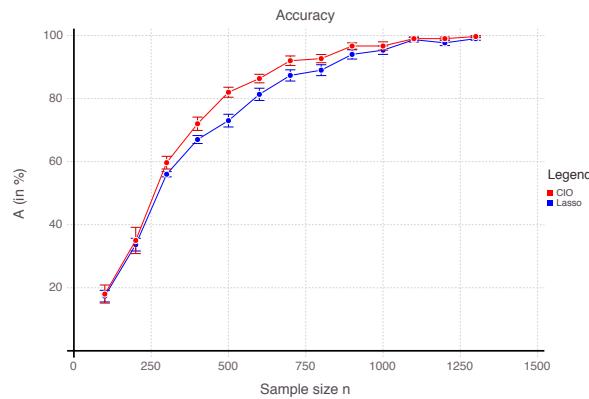
$$F(\beta) = |\{j : w_j \neq 0, w_{true,j} = 0\}| \in \{0, \dots, p\}.$$

Obviously, $A(\beta) + F(\beta) = |\{j : w_j \neq 0\}| = k$. A classifier β is said to *perfectly recover* the true support if it selects the truth ($A(\beta) = k_{true}$) and nothing but the truth ($F(\beta) = 0$ or equivalently $k = k_{true}$).

Selecting the truth...

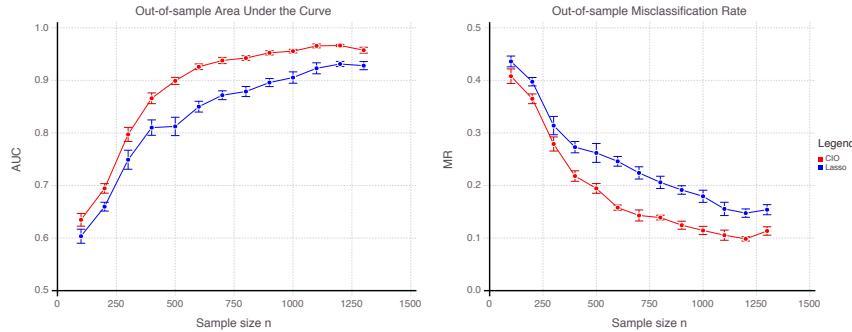
We first compare the performance of Algorithm 6.1 for sparse logistic regression with a Lasso logistic regression, when both methods are given the true number of features in the support k_{true} . A key property in this context for any best subset selection method, is that it selects the true support as sample size increases, as represented in Figure 6.1. From that perspective, both methods demonstrate a similar convergence: As n increases, both classifiers end up selecting the truth, with Algorithm 6.1 needing somewhat smaller number of samples than Lasso.

Figure 6.1: Evolution of the number of true features selected as sample size n increases, for Lasso (in blue) and sparse logistic regression (in red). Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing n from 100 to 1,300.



Apart from the absolute number of true/false features selected, one might wonder whether the features selected are actually good features in terms of predictive power. In this metric, sparse logistic regression significantly outperforms the Lasso classifier, both in terms of Area Under the Curve (AUC) and misclassification rate, as shown on Figure 6.2, demonstrating a clear predictive edge of exact sparse formulation.

Figure 6.2: Evolution of AUC (left) and misclassification rate (right) on a validation set as sample size n increases, for Lasso (in blue) and sparse logistic regression (in red). Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing n from 100 to 1,300.



In terms of computational complexity Figure 6.3 represents the number of cuts (left panel) and computational time (right panel) required by the cutting-plane algorithm as the problem size n increases for a fixed value of $\gamma \times n$. For low values of n , the number of cuts is in the thousands. Quite surprisingly, it does not increase with n . On the contrary, having more observations reduces the number of cuts down to less than twenty. Computational time evolves similarly: the algorithm reaches the time limit (here, 3 minutes) when n is low, but terminates in a few seconds for high values. For sparse linear regression, in Chapter 3 we observed a similar, yet even sharper, phase transition in computational complexity. The threshold value, however, increases with γ . In other words, when n is fixed, computational time increases as γ increases, which corroborates the intuition that in the limit $\gamma \rightarrow 0$, $\beta^* = 0$ is obviously optimal, while the problem can be ill-posed as $\gamma \rightarrow +\infty$. Since the right regularization parameter is unknown *a priori*, one needs to test high but sometimes relevant values of γ for which the time limit is reached and the overall procedures terminates in minutes, while GLMNet requires less than a second. As for the choice of the time limit, it does not significantly impact the performance of the algorithm: As shown on Figure 6.4, the algorithm quickly finds the optimal solution and much of the computational time is

spent improving on the lower bound, i.e., proving the solution is indeed optimal.

Figure 6.3: Evolution of the number of cuts (left panel) and computational time (right panel) required by the outer-approximation algorithm as sample size n increases, for sparse logistic regression. Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$, $\gamma = \gamma_0/n$ with $\gamma_0 = 2^5 p / (k \max_i \|\mathbf{x}_i\|^2)$ and increasing n from 100 to 1,300.

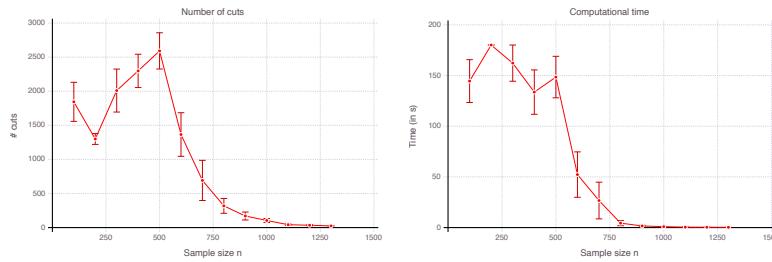
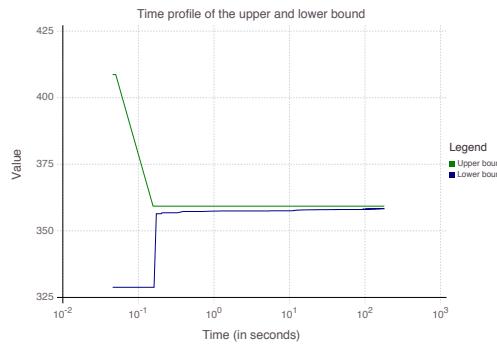


Figure 6.4: Evolution of the upper (best feasible solution, in green) and lower bounds (in blue) in Algorithm 6.1 as computational time (in log scale) increases, for one problem instance with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$, $\gamma = \gamma_0/n$ with $\gamma_0 = 2^7 p / (k \max_i \|\mathbf{x}_i\|^2)$ and $n = 600$.

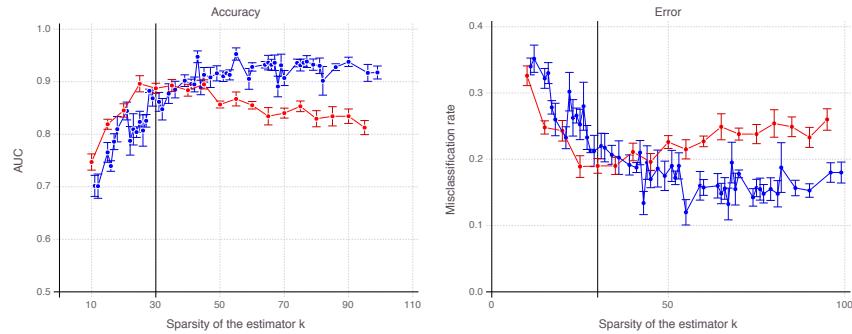


...and nothing but the truth

In practice, however, the length of the true support k_{true} is unknown *a priori* and is to be determined using cross-validation. Given a data set with a fixed number of samples n and features p , we compute classifiers with different values of sparsity parameter k and choose the value which

leads to the best accuracy on a validation set. Irrespective of the method, AUC as a function of sparsity k should have an inverted-U shape: if k is too small, not enough features are taken into account to provide accurate predictions. If k is too big, the model is too complex and overfits the training data. Hence, there is some optimal value k^* which maximizes validation AUC.¹ Figure 6.5 represents the evolution of both the AUC and misclassification rate on a validation set as sparsity k increases for Lasso and the exact sparse logistic regression. The exact CIO formulation leads to an optimal sparsity value k_{CIO}^* which is much closer to the truth than k_{Lasso}^* , as shown on the left panel of Figure 6.6. In addition, Figure 6.6 also exposes a major deficiency of Lasso as a feature selection method: even when the number of samples increases, Lasso fails to select the relevant features *only* and returns a support k_{Lasso}^* much larger than the truth whereas k_{CIO}^* converges to k_{true} quickly as n increases, hence selecting the truth and nothing but the truth.

Figure 6.5: Evolution of validation AUC (left) and misclassification rate (right) as sparsity of the classifier k increases, for Lasso (in blue) and sparse logistic regression (in red). Results correspond to average values obtained over 10 data sets with $n = 700$, $p = 1,000$, $k_{true} = 30$ (black vertical line), $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing k from 0 to 100.

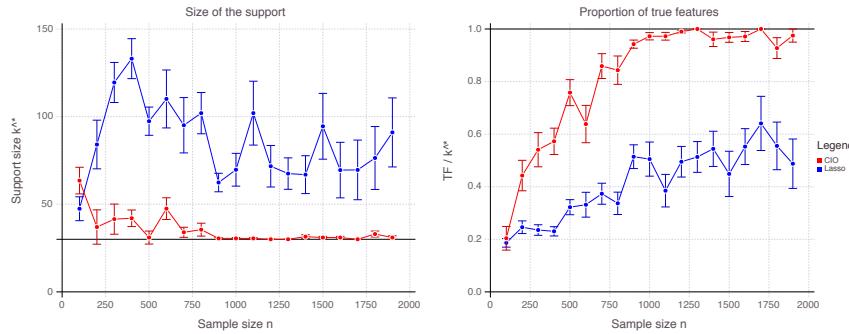


Real-world data sets

We now apply the sparse classification algorithm on real-world data, of various size and dimensions.

¹Equivalently, one could use misclassification rate instead of AUC as a performance metric.

Figure 6.6: Evolution of optimal sparsity k^* (left) and proportion of true features TF/k^* (right) as sample size n increases, for Lasso (in blue) and sparse logistic regression (in red). Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing n from 100 to 1,900.



Over-determined regime $n > p$

We consider data sets from the UC Irvine Machine Learning Repository, split them into a training and a test set (80%/20%), calibrate a sparse and a Lasso classifier on the training data, using cross-validation to fit the hyper-parameters k and γ in the sparse case and λ in the Lasso case, and compare AUC and misclassification rate on the test set for both methods. Characteristics of these data sets and experimental results are given in Table 6.2. Experiments clearly demonstrate that the sparse formulation scales to data sets of these sizes while delivering predictions similar to Lasso with classifiers slightly sparser, suggesting features selected are more statistically relevant. Yet, these data sets are not very insightful for subset selection methods because of the limited number of features p .

Table 6.2: Comparative results of Lasso and sparse logistic regression on data sets from UCI ML Repository.

Data set	n	p	Sparsity k		Misclassif.		AUC	
			Spar.	Las.	Spar.	Las.	Spar.	Las.
Bank. Auth.	1,372	5	3	3	4.01%	4.01%	0.999	0.998
Br. Canc.	683	10	6	7	1.47%	2.21%	0.993	0.994
Br. Canc. D.	569	31	3	10	6.19%	2.65%	0.990	0.996
Chess	3,196	38	36	36	2.82%	2.66%	0.996	0.996
Magic Tel.	19,020	11	6	8	20.11%	21.90%	0.840	0.834
QSAR Biod.	1,055	42	40	37	13.27%	21.8%	0.914	0.909
Spambase	4,601	58	55	57	7.61%	6.20%	0.970	0.976

6.4 Sparse Support Vector Machine

In the Support Vector Machine (SVM) framework, we consider the case of the hinge loss

$$\ell(y, u) = \max(0, 1 - yu),$$

whose Fenchel conjugate is

$$\hat{\ell}(y, \alpha) = \max_{u \in \mathbb{R}} u\alpha - \max(0, 1 - yu) = \begin{cases} y\alpha, & \text{if } y\alpha \in [-1, 0], \\ +\infty, & \text{otherwise.} \end{cases}$$

Our framework can therefore be applied, the conditions $y_i\alpha_i \in [-1, 0]$ being implemented as additional cuts in Algorithm 6.1.

Regarding the ℓ_1 -regularized SVM formulation:

$$\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^n \max \left(0, 1 - y_i \beta^T \mathbf{x}_i \right) + \lambda \|\beta\|_1 \quad (6.9)$$

we introduce slack variables ξ_i to encode for $\max(0, 1 - y_i \beta^T \mathbf{x}_i)$ and solve it as a linear optimization problem.

Simulations on synthetic data

We first assess the sparse SVM on synthetic data sets and compare it to a state-of-the-art Lasso procedure.

We generate data according to the same methodology as for the logistic regression case in Section 6.3, except that we generate y_i as follows:

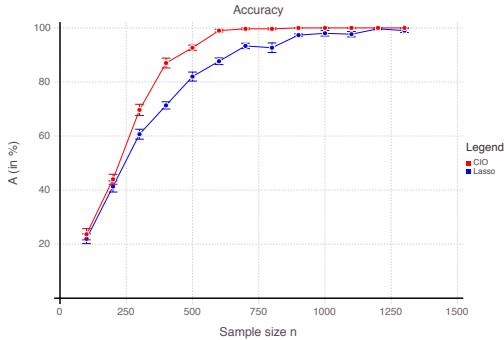
$$y_i = \begin{cases} 1, & \text{if } \beta_{true}^T \mathbf{x}_i + \varepsilon_i > 0, \\ -1, & \text{otherwise.} \end{cases}$$

Selecting the truth...

Given the true support size k_{true} , Figure 6.7 represents how the number of true features evolves as n increases. Both Lasso and sparse SVM demonstrate asymptotic recovery and $TF \rightarrow k_{true}$ as sample size tends to infinity. In addition to selecting notably more correct features (Figure 6.7), the sparse SVM formulation also has a clear advantage in terms of predictive power, as shown on Figure 6.8.

In terms of computational time, Figure 6.9 represents the number of cuts (left panel) and computational time (right panel) required by Algorithm 6.1 as the problem size n increases. The overall behavior is similar to the case of the logistic loss observed in Figure 6.3. However, the number of cuts required in the case of the hinge loss is three times less than for logistic regression.

Figure 6.7: Evolution of the number of true features selected as sample size n increases, for Lasso (in blue) and sparse SVM (in red). Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing n from 100 to 1,300.



...and nothing but the truth

In real-life applications however, true support size needs to be estimated using cross-validation. Figure 6.11 represents the support size k^* as well as the proportion of true features TF/k^* for both formulations. Similarly to the logistic regression case, Lasso SVM does not appear as a satisfying subset selection method since k_{Lasso}^* does not converge to k^{true} when $n \rightarrow \infty$. On the other hand, Sparse SVM is very efficient in selecting the right number of features, even with a moderate number of samples.

Real-world data sets

Over-determined regime $n > p$

Table 6.3 compiles characteristics of the UCI ML Repository data sets and experimental results for Lasso and Sparse SVM. These data sets are not very insightful because of the limited number of features p but they demonstrate that (a) the exact sparse formulation scales to problems of such size, (b) exact sparsity provides sparser classifiers than Lasso without compromising on the prediction accuracy.

Under-determined regime $p > n$

In the highly under-determined regime encountered in the Lung Cancer data (Table 6.4), sparse SVM returns 30 relevant genes only, far less than Lasso SVM, while outperforming in terms of AUC.

Figure 6.8: Evolution of AUC (left) and misclassification rate (right) on a validation set as sample size n increases, for Lasso (in blue) and sparse SVM (in red). Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing n from 100 to 1,300.

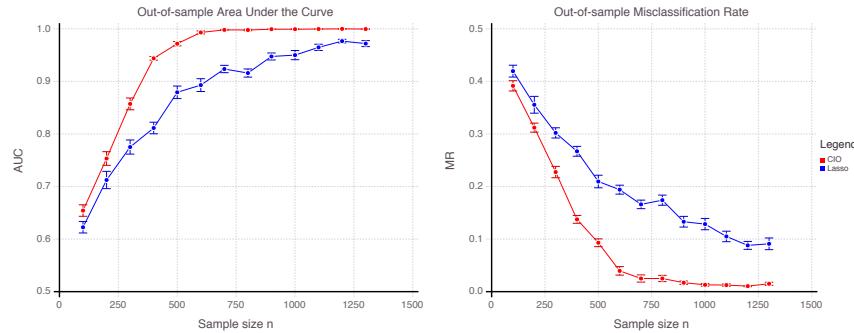


Table 6.3: Comparative results of Lasso and Sparse SVM on data sets from UCI ML Repository.

Data set	n	p	Sparsity k		Misclassif.		AUC	
			Spar.	Las.	Spar.	Las.	Spar.	Las.
Bank. Auth.	1,372	4	4	4	1.10%	1.83%	0.999	0.999
Br. Canc.	683	9	6	9	2.94%	2.94%	0.995	0.993
Br. Canc. D.	569	30	3	9	4.42%	7.96%	0.960	0.956
Chess	3,196	37	36	35	2.03%	1.56%	0.997	0.997
Magic Tel.	19,020	10	9	10	20.61%	20.58%	0.843	0.843
QSAR Biod.	1,055	41	30	29	13.74%	12.32%	0.939	0.942
Spambase	4,601	57	55	57	6.52%	6.63%	0.978	0.977

6.5 Concluding Remarks

In this chapter, we have proposed a tractable binary convex optimization algorithm for solving sparse classification. Though theoretically NP-hard, the algorithm scales for logistic regression and SVM in problems with n, p in 10,000s within minutes. Comparing the method and Lasso, we observe empirically that as n increases, the number of true features selected by both methods converges to the true sparsity. Apart from accuracy, the exact sparse formulation has an edge over Lasso in the number of false features: as n increases, the number of false features selected by Algorithm 6.1 converges to zero, while this is not observed for Lasso. This phenomenon is also observed for classifying the type of cancer using gene expression data from the Cancer Genome Atlas Research Network with $n = 1,145$ lung cancer patients and $p = 14,858$ genes. Sparse classification using logistic

Figure 6.9: Evolution of the number of cuts (left panel) and computational time (right panel) required by the outer-approximation algorithm as sample size n increases for sparse SVM. Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$, $\rho = 0.3$, $SNR \rightarrow \infty$, $\gamma = \gamma_0/n$ with $\gamma_0 = 2^5 p / (k \max_i \| \mathbf{x}_i \|^2)$ and increasing n from 100 to 1,300.

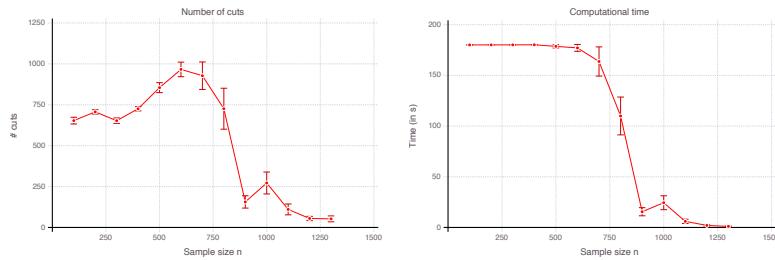


Table 6.4: Comparative results of Lasso and Sparse SVM on Lung Cancer data.

Data set	n	p	Sparsity k		AUC	
			Sparse	Lasso	Sparse	Lasso
Lung cancer	1,145	14,858	30	172	0.9750	0.9697

regression returns a classifier based on 50 genes versus 171 for Lasso and using SVM 30 genes versus 172 for Lasso with similar predictive accuracy.

6.6 Notes and Sources

The material for this chapter is from [42].

Figure 6.10: Evolution of validation AUC (left) and misclassification rate (right) as sparsity of the classifier k increases, for Lasso (in blue) and sparse (in red) SVM. Results correspond to average values obtained over 10 data sets with $n = 700$, $p = 1,000$, $k_{true} = 30$ (black vertical line), $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing k from 0 to 100.

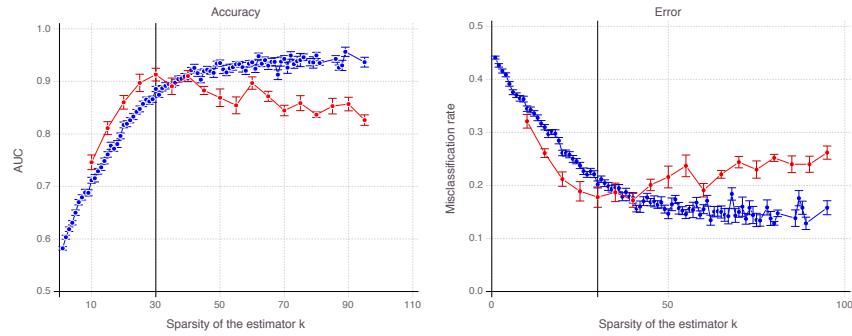
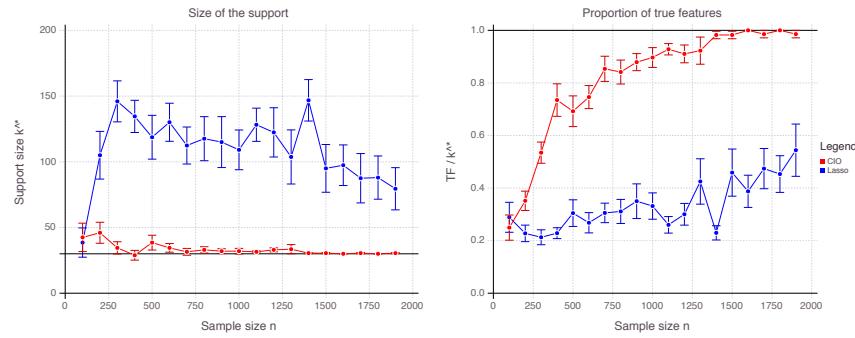


Figure 6.11: Evolution of optimal sparsity k^* (left) and proportion of true features TF/k^* (right) as sample size n increases, for Lasso (in blue) and sparse SVM (in red). Results correspond to average values obtained over 10 data sets with $p = 1,000$, $k_{true} = 30$ (black lines), $\rho = 0.3$, $SNR \rightarrow \infty$ and increasing n from 100 to 1,900.



Part II

**Optimal Trees for
Classification and
Regression**

Chapter 7

Classification and Regression Trees

This is the essence of intuitive heuristics: when faced with a difficult question, we often answer an easier one instead, usually without noticing the substitution.

– Daniel Kahneman

Contents

- 7.1. Overview of CART
- 7.2. Limitations of CART
- 7.3. Random Forests and Boosted Trees
- 7.4. Notes and Sources

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. [54]. 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, present random forests and boosted trees.

7.1 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 7.1 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 [172]. 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 7.1a, we plot the data plotted according to the petal length and width. Figures 7.1b–7.1e 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 7.1f 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 7.2.

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 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 7.1, we see that the fourth split, applied in Figure 7.1e, 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

Figure 7.1: An example of the CART algorithm applied to the “Iris” dataset that shows the recursive partitioning step-by-step.

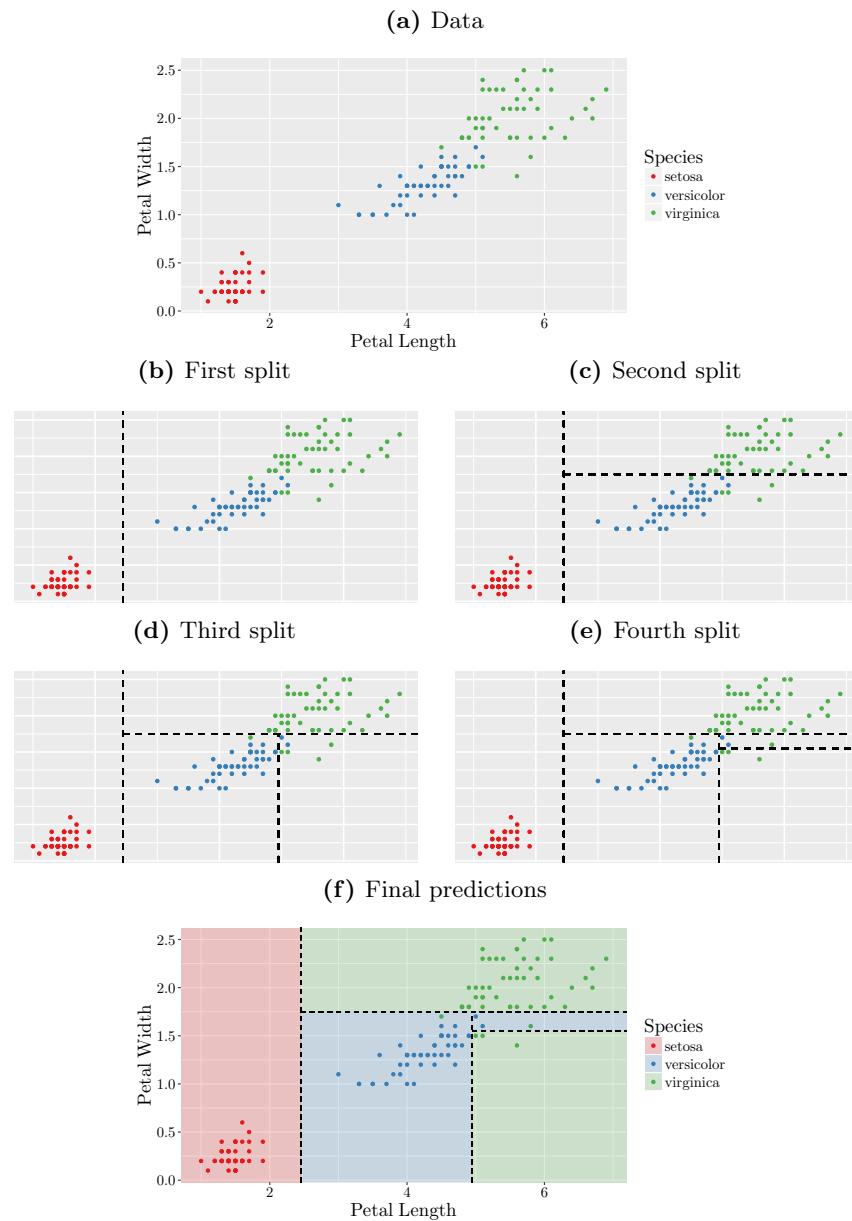
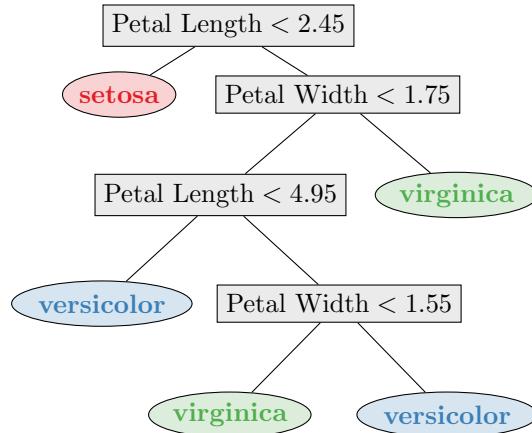
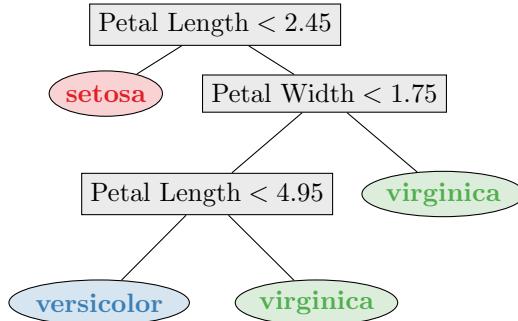
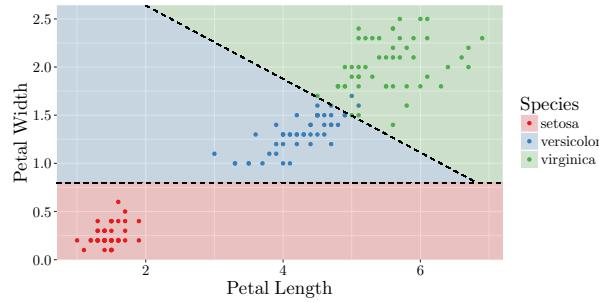


Figure 7.2: The tree learned by CART for the “Iris” dataset.**Figure 7.3:** The CART tree for the “Iris” dataset after conducting pruning.

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 7.2, and if we prune this tree with the complexity parameter $\alpha = 0.01$, we get the tree shown in Figure 7.3. 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 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

Figure 7.4: An example of partitioning the “Iris” dataset with arbitrary hyperplanes rather than just splits parallel to the axes.



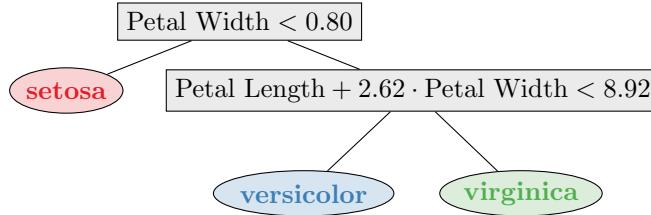
tuning the value of α for CART is known as *cost-complexity pruning*, which we will introduce in Section 8.4.

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 8, we can verify that the tree in Figure 7.3 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 7.1, 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 7.4, and the corresponding tree that generates these partitions is given in Figure 7.5. 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 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.

7.2 Limitations of CART

The main shortcoming of the top-down approach taken by CART and other popular decision tree methods like C4.5 [220] and ID3 [219], is its

Figure 7.5: A tree for the “Iris” dataset with hyperplane splits.

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 7.6 shows an example where a tree grown via top-down induction is very different from the true tree that generated the data.

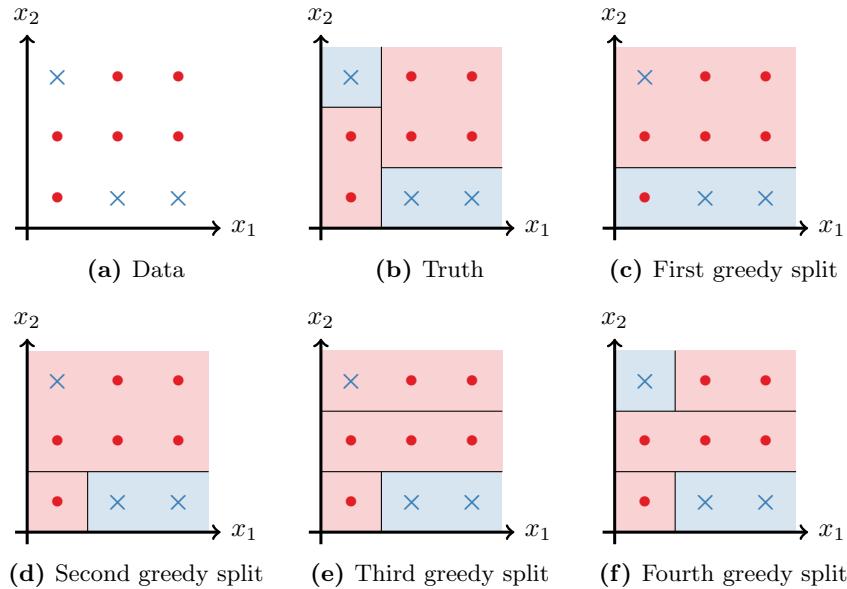
Another limitation of top-down induction methods is that they typically require pruning to achieve trees that generalize well. Pruning is needed because top-down induction is unable to handle penalties (we call them complexity penalties) on how large the tree becomes while growing it, since powerful 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 [207], LSID3 and ID3-k [94] 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 [199].

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. [54, 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

Figure 7.6: An example where greedy tree induction fails to learn the truth in the data. Figure 7.6a shows the data, and the true decision tree that generated the data is given in Figure 7.6b. Figures 7.6c–7.6f 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.



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 7.6, where adding a split to Figure 7.6d to get to Figure 7.6e 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. [54, 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* [146]. 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 [14], continuous optimization [15], dynamic programming [76, 215], genetic algorithms [244], and more recently, evolutionary algorithms [123], and optimizing an upper bound on the tree error using stochastic gradient descent [206]. 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 [5], T3 [253] and T3C [261], 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 [253, 261].

7.3 Random Forests and Boosted Trees

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+” [59]. 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 [58], 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 [103]. 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 [72].

The Accuracy-Interpretability Tradeoff

Although both of these methods achieve state-of-the-art accuracies, note that they are ensemble methods, meaning they average a large collection of trees and make predictions as a black-box, rather than generating a single decision tree like CART. The interpretability of these methods is therefore vastly diminished. Breiman himself remarked [59]:

So forests are A+ predictors. But their mechanism for producing a prediction is difficult to understand. Trying to delve into the tangled web that generated a plurality vote from 100 trees is a Herculean task. So on interpretability, they rate an F.

This leads to the current dilemma that machine learning practitioners 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 the Optimal Trees approach we present in this part of the book can allow us to do away with this tradeoff entirely. Our goal is for the 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.

7.4 Notes and Sources

Leo Breiman et. al. [54] proposed CART. This book has 38,216 Google Scholar citations (as of September, 2018) in google scholar, and it is one of the most cited works in the mathematical sciences. Leo Breiman [58] proposed random forests. The paper has 38,817 citations (as of September, 2018) in google scholar. The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an

optimization algorithm on a suitable cost function [53]. Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman [103] simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean [?]. The latter two papers introduced the view of boosting algorithms as iterative functional gradient descent algorithms. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction.

Chapter 8

Optimal Classification Trees with Parallel Splits

On interpretability, trees rate an A+.

– Leo Breiman

Contents

- 8.1. Review of Classification Tree Methods
- 8.2. A Mixed-Integer Optimization Approach
- 8.3. A Local Search Approach
- 8.4. A Method for Tuning Hyperparameters
- 8.5. Experiments with Synthetic Datasets
- 8.6. Experiments with Real-World Datasets
- 8.7. Concluding Remarks
- 8.8. Notes and Sources

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 [34], 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 book 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.

8.1 Review of Classification Tree Methods

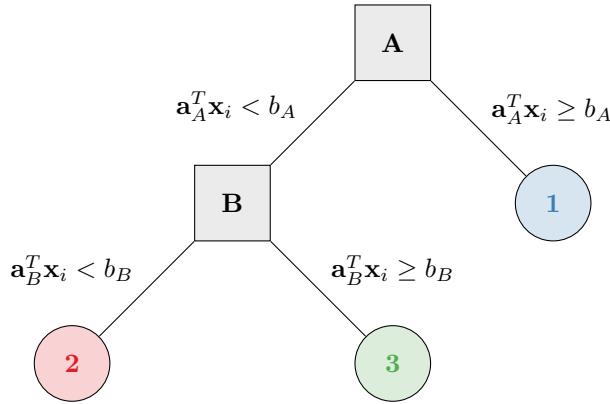
We are given the training data (\mathbf{X}, \mathbf{y}) , containing n observations (\mathbf{x}_i, y_i) , $i \in [n]$, each with p features $\mathbf{x}_i \in \mathbb{R}^p$ and a class label $y_i \in [K]$ indicating

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 $0\text{--}1$ 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 8.1. 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 8.1: An example of a decision tree with two branch nodes, A and B, and three leaf nodes, 1, 2 and 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 8.2

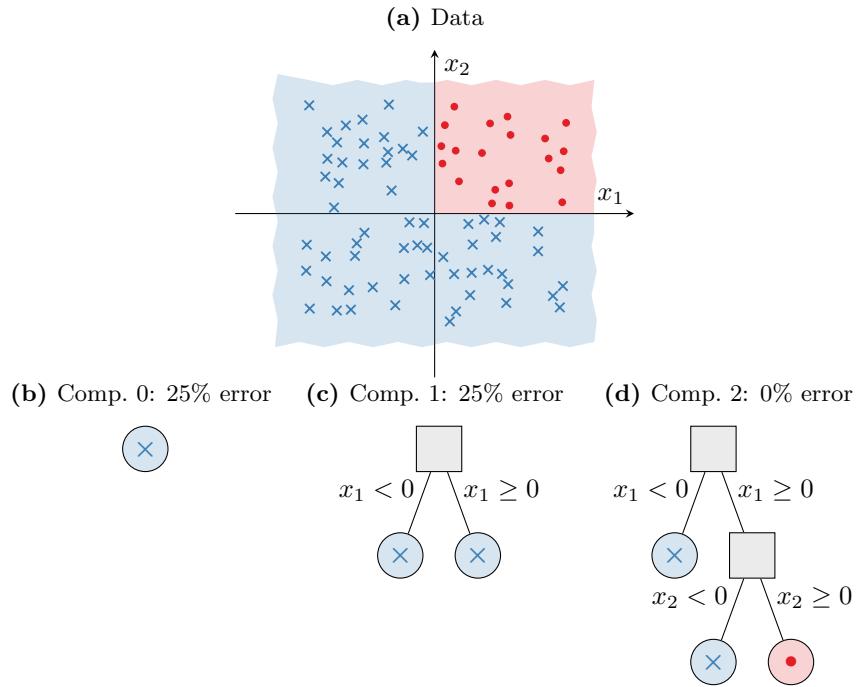
Using the details of the CART procedure, we can state the problem that CART attempts to solve as a formal optimization problem. There are two parameters in this problem. The tradeoff between accuracy and complexity of the tree is controlled by the complexity parameter α , and the minimum number of points 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:

$$\begin{aligned} \min \quad & R(\mathbb{T}) + \alpha|\mathbb{T}| \\ \text{s.t. } & N(\ell) \geq N_{\min} \quad \forall \ell \in \text{leaves}(\mathbb{T}) \end{aligned} \tag{8.1}$$

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 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.

Figure 8.2: 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 8.2a is generated according to a simple AND boolean rule: points in the positive quadrant have label \bullet , otherwise they are labeled \times . Figures 8.2b–8.2d 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.



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 [200] 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.

8.2 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 (8.1) is using MIO, we note that at every step in tree creation, we are required to make a number of discrete decisions:

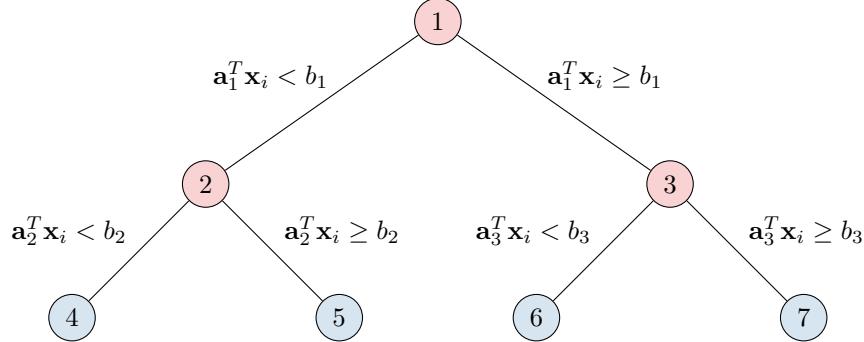
- At every new node, we must choose to either branch or stop.
- After choosing to stop branching at a node, we must choose a label to assign to this new leaf node.
- After choosing to branch, we must choose which of the variables to branch on.
- When classifying the training points according to the tree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

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 (8.1) 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 8.3 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

Figure 8.3: The maximal tree for depth $D = 2$. Branch nodes are red and leaf nodes are blue.



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 8.3, $A_L(5) = \{1\}$, $A_R(5) = \{2\}$, and $A(5) = \{1, 2\}$.

We divide the nodes in the tree into two sets:

Branch nodes: Nodes $t \in \mathcal{T}_B = \{1, \dots, \lfloor T/2 \rfloor\}$ apply a split of the form $\mathbf{a}^T \mathbf{x} < b$. 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:

$$\sum_{j=1}^p a_{jt} = d_t, \quad \forall t \in \mathcal{T}_B, \tag{8.2}$$

$$0 \leq b_t \leq d_t, \quad \forall t \in \mathcal{T}_B, \tag{8.3}$$

$$a_{jt} \in \{0, 1\}, \quad \forall j \in [p], t \in \mathcal{T}_B, \quad (8.4)$$

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 $0 \leq \mathbf{a}_t^\top \mathbf{x}_i \leq d_t$ 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.

$$d_t \leq d_{p(t)}, \quad \forall t \in \mathcal{T}_B \setminus \{1\}, \quad (8.5)$$

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} :

$$z_{it} \leq l_t, \quad \forall t \in \mathcal{T}_B, \quad (8.6)$$

$$\sum_{i=1}^n z_{it} \geq N_{\min} l_t, \quad \forall t \in \mathcal{T}_B. \quad (8.7)$$

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

$$\sum_{t \in \mathcal{T}_L} z_{it} = 1, \quad \forall i \in [n]. \quad (8.8)$$

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

$$\mathbf{a}_m^\top \mathbf{x}_i < b_m + M_1(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \quad (8.9)$$

$$\mathbf{a}_m^\top \mathbf{x}_i \geq b_m - M_2(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \quad (8.10)$$

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 (8.9) 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 (8.9) and change the inequality to be non-strict:

$$\mathbf{a}_m^\top \mathbf{x}_i + \epsilon \leq b_m + M_1(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t). \quad (8.11)$$

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 j th feature and take

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

where $x_j^{(i)}$ is the i th largest value in the j th 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:

$$\mathbf{a}_m^T(\mathbf{x}_i + \epsilon) \leq b_m + M_1(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t). \quad (8.12)$$

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 $b_t - \mathbf{a}_t^T \mathbf{x}_i$ is 1, and so we can set $M_2 = 1$. This gives the following final constraints that will enforce the splits in the tree: too long

$$\mathbf{a}_m^T(\mathbf{x}_i + \epsilon) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \quad (8.13)$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - (1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t). \quad (8.14)$$

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 :

$$N_{kt} = \sum_{i: y_i=k} z_{it}, \quad \forall t \in \mathcal{T}_L, k \in [K], \quad (8.15)$$

$$N_t = \sum_{i=1}^n z_{it}, \quad \forall t \in \mathcal{T}_L. \quad (8.16)$$

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

$$c_t = \arg \max_{k \in [K]} \{N_{kt}\}. \quad (8.17)$$

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:

$$\sum_{k=1}^K c_{kt} = l_t, \quad \forall t \in \mathcal{T}_L. \quad (8.18)$$

Since we know how to make the optimal prediction at each leaf t using (8.17), 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:

$$L_t = N_t - \max_{k \in [K]} \{N_{kt}\} = \min_{k \in [K]} \{N_t - N_{kt}\}, \quad (8.19)$$

which can be linearized to give

$$L_t \geq N_t - N_{kt} - M(1 - c_{kt}), \quad \forall t \in \mathcal{T}_L, k \in [K], \quad (8.20)$$

$$L_t \leq N_t - N_{kt} + Mc_{kt}, \quad \forall t \in \mathcal{T}_L, k \in [K], \quad (8.21)$$

$$L_t \geq 0, \quad \forall t \in \mathcal{T}_L, \quad (8.22)$$

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 (8.1) can be written:

$$\min \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \cdot C. \quad (8.23)$$

Putting all of this together gives the following MIO formulation for problem (8.1), which we call the *Optimal Classification Trees* (OCT) model:

$$\min \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \cdot C \quad (8.24)$$

$$\text{s.t. } L_t \geq N_t - N_{kt} - n(1 - c_{kt}), \quad \forall t \in \mathcal{T}_L, k \in [K],$$

$$L_t \leq N_t - N_{kt} + nc_{kt}, \quad \forall t \in \mathcal{T}_L, k \in [K],$$

$$L_t \geq 0, \quad \forall t \in \mathcal{T}_L,$$

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

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

$$\begin{aligned}
& \sum_{k=1}^K c_{kt} = l_t, & \forall t \in \mathcal{T}_L, \\
& C = \sum_{t \in \mathcal{T}_B} d_t, \\
& \mathbf{a}_m^\top \mathbf{x}_i \geq b_m - (1 - z_{it}), & \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \\
& \mathbf{a}_m^\top (\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), & \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \\
& \sum_{t \in \mathcal{T}_L} z_{it} = 1, & \forall i \in [n], \\
& z_{it} \leq l_t, & \forall t \in \mathcal{T}_L, \\
& \sum_{i=1}^n z_{it} \geq N_{\min} l_t, & \forall t \in \mathcal{T}_L, \\
& \sum_{j=1}^p a_{jt} = d_t, & \forall t \in \mathcal{T}_B, \\
& 0 \leq b_t \leq d_t, & \forall t \in \mathcal{T}_B, \\
& d_t \leq d_{p(t)}, & \forall t \in \mathcal{T}_B \setminus \{1\}, \\
& z_{it}, l_t, c_{kt} \in \{0, 1\}, & \forall i \in [n], k \in [K], t \in \mathcal{T}_L, \\
& a_{jt}, d_t \in \{0, 1\}, & \forall j \in [p], t \in \mathcal{T}_B.
\end{aligned}$$

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.

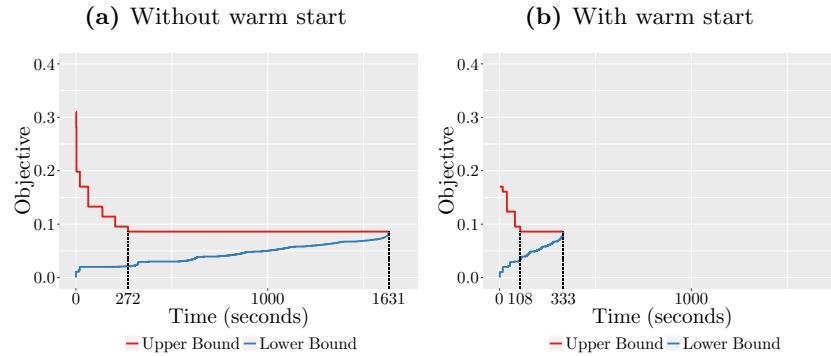
There are three hyper-parameters that need to be specified: the maximum depth D , the minimum leaf size N_{\min} , and the complexity parameter α .

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 [36]. 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 (8.24); we can use CART to generate these warm start solutions.

Figure 8.4: Comparison of upper and lower bound evolution while solving MIO problem (8.24) with and without warm starts for a tree of depth $D = 2$ for the Wine dataset with $n = 178$ and $p = 13$.



Given a solution from CART, it is simple to construct a corresponding feasible solution to the MIO problem (8.24) using the splits from CART to infer the values of the remaining variables in the MIO problem.

By design, the parameters N_{\min} and α have the same meaning in CART and in problem (8.24), 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 8.4 shows an example of the typical evolution of upper and lower bounds as we solve the MIO problem (8.24) 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.

8.3 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.

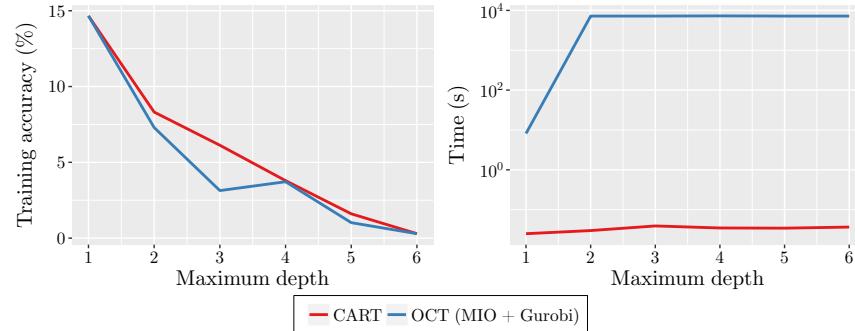
Limitations of the MIO approach

The MIO formulation for the Optimal Classification Trees problems as presented in Section 8.2 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 [172]. 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 (8.1). We solved the OCT MIO problem using the Gurobi solver [125], 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 8.5 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

Figure 8.5: Training accuracy (%) and running time for each method on “Banknote Authentication” dataset.



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 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 $\mathbf{A} = \{\mathbf{a}_1, \dots, \mathbf{a}_{\lfloor T/2 \rfloor}\}$ and $\mathbf{b} = \{b_1, \dots, b_{\lfloor T/2 \rfloor}\}$. 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:

$$\min_{\mathbf{A}, \mathbf{b}} \text{error}(\mathbf{A}, \mathbf{b}, \mathbf{X}, \mathbf{y}), \quad (8.25)$$

where $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ and $\mathbf{y} = \{y_1, \dots, y_n\}$ 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 (8.25) is greatly reduced compared to problem (8.24); 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 (8.25). 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 t th 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} ;

Algorithm 8.1 LOCALSEARCH**Input:** Starting decision tree \mathbb{T} ; training data \mathbf{X}, \mathbf{y} **Output:** Locally optimal decision tree

```

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

```

- $\text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$ 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} .

For Optimal Classification Trees, the loss function corresponds to the objective of Problem (8.24), which we can write as:

$$\text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y}) = \frac{1}{\hat{L}} L(\mathbb{T}, \mathbf{X}, \mathbf{y}) + \alpha \cdot \text{complexity}(\mathbb{T}), \quad (8.26)$$

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 $\text{complexity}(\mathbb{T})$ 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 (8.25) with respect to our search procedure, so we stop and begin the procedure again with a new tree. This procedure is detailed in Algorithm 8.1, which depends on the supplementary functions given in Algorithms 8.2 and 8.3.

The local search procedure given in Algorithm 8.1 depends on the function `OPTIMIZE NODE PARALLEL` in Algorithm 8.2, 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 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

Algorithm 8.2 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:    $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}} \leftarrow \text{new leaf nodes}$ 
5:    $\text{error}_{\text{best}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$                                 ▷ error of current root
6:
7:    $\mathbb{T}_{\text{para}}, \text{error}_{\text{para}} \leftarrow \text{BESTPARALLELSPLIT}(\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}, \mathbf{X}, \mathbf{y})$ 
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  $\mathbb{T}$ 
```

the new loss after performing each of these three substitutions 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 8.3 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.

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 [58]. 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

Algorithm 8.3 BESTPARALLELSPLIT

Input: Lower and upper subtrees T_{lower} and T_{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  $j = 1, \dots, p$  do                                 $\triangleright \text{loop over all dimensions}$ 
4:    $\text{values} \leftarrow \{X_{ij} : i = 1, \dots, n\}$ 
5:   sort values in ascending order
6:   for  $i = 1, \dots, n - 1$  do                   $\triangleright \text{loop over all split placements}$ 
7:      $b \leftarrow \frac{1}{2}(\text{values}_i + \text{values}_{i+1})$ 
8:      $T \leftarrow$  branch node  $\mathbf{e}_j^T \mathbf{x} < b$  with children  $T_{\text{lower}}, T_{\text{upper}}$ 
9:     if  $\text{minleafsize}(T) \geq N_{\text{min}}$  then           $\triangleright \text{check feasibility}$ 
10:       $\text{error} \leftarrow \text{loss}(T, \mathbf{X}, \mathbf{y})$ 
11:      if  $\text{error} < \text{error}_{\text{best}}$  then           $\triangleright \text{save split if better}$ 
12:         $\text{error}_{\text{best}} \leftarrow \text{error}$ 
13:         $T_{\text{best}} \leftarrow T$ 
14: return  $T_{\text{best}}, \text{error}_{\text{best}}$ 
```

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 8.3. 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 8.2. 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 8.1. 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 $O(n^2 pKT)$. 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 $O(n^2 pKT)$ for the local search. This final step dominates giving a total cost of $O(n^2 pKT)$. 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 $O(n^3 pK)$. Under this worst case, the cost of CART would be $O(n^2 pK)$, and so the additional power of OCT increases the cost simply by a factor of n .

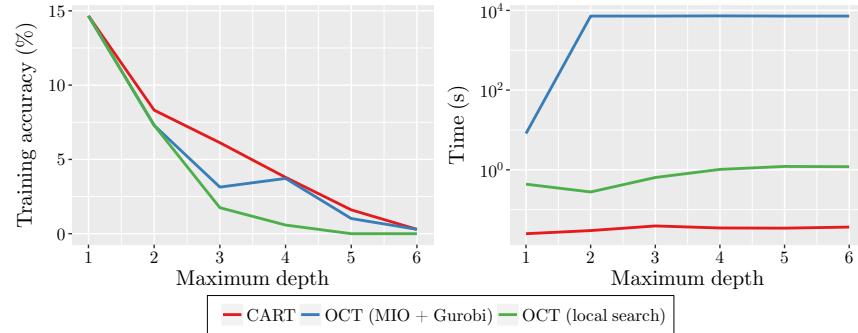
A set of more realistic assumptions for practical use is that the number of nodes in the tree would be $O(\log n)$ and the number of local search iterations required is also $O(\log n)$. Under this scenario, the cost of OCT is $O(npK \log^2 n)$ and the cost of CART is $O(npK \log n)$, and so OCT is only 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 Authentication” dataset. Figure 8.6 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, as there is clearly great room for improvement. Finally, it takes around

figure out how to prove when non-zero, complexity is integral-add/remove/reopt

Figure 8.6: Training accuracy (%) and running time for each method on “Banknote Authentication” dataset.



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.

8.4 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 [54]. 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 t th

Algorithm 8.4 PRUNE

Input: Starting decision tree \mathbb{T}

Output: Vector α of critical values for complexity parameter; vector 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  $t \in \text{branches}(\mathbb{T})$  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  $\alpha, v$ 

```

branch node, $\text{complexity}(\mathbb{T}_t)$ be the number of splits in the subtree rooted at the t th branch node, and $R(t)$ be the misclassification error that would be realized if the t th branch node was replaced with a leaf. The change in objective function by replacing the t th branch node with a leaf is therefore

$$\underbrace{R(t)}_{\text{leaf error}} - \underbrace{[R(\mathbb{T}_t) + \alpha \cdot \text{complexity}(\mathbb{T}_t)]}_{\text{branch error}}. \quad (8.27)$$

This substitution will improve the objective if (8.27) is negative, or equivalently if

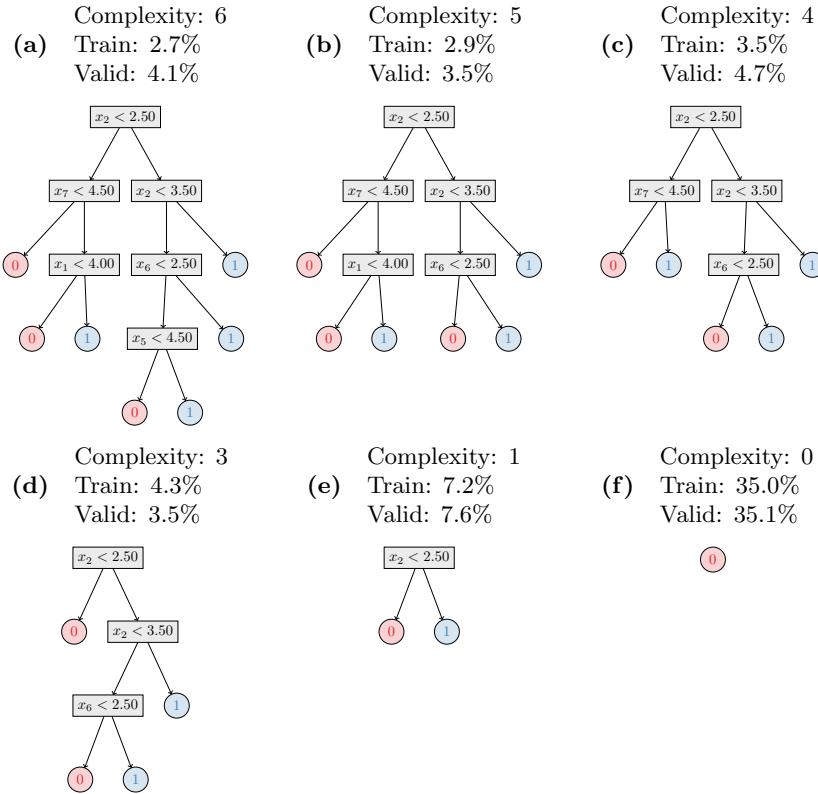
$$\alpha > \frac{R(t) - R(\mathbb{T}_t)}{\text{complexity}(\mathbb{T}_t)} \triangleq g(t). \quad (8.28)$$

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 α . We therefore identify $\hat{t} = \arg \min_t g(t)$ 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 8.4.

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 8.5 describes this process.

Figure 8.7 demonstrates the cost-complexity pruning process as applied to a tree trained on the “Breast Cancer Wisconsin” dataset from the UCI Machine Learning Repository [172].

Figure 8.7: Example of pruning process following Algorithm 8.4 on the “Breast Cancer Wisconsin” dataset. The starting tree in Figure 8.7a 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 (8.28) 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 8.5.



Algorithm 8.5 TUNECP**Input:** Starting decision tree \mathbb{T} **Output:** Tuned value of α that minimizes validation error

- | | |
|--|---|
| 1: $\boldsymbol{\alpha}, \mathbf{v} \leftarrow \text{PRUNE}(\mathbb{T})$ | \triangleright <i>Generate sequence of pruned trees</i> |
| 2: $i \leftarrow \arg \min_j v_j$ | \triangleright <i>Find interval with minimal error</i> |
| 3: return $\frac{1}{2}(\alpha_i + \alpha_{i+1})$ | \triangleright <i>Return midpoint of interval</i> |

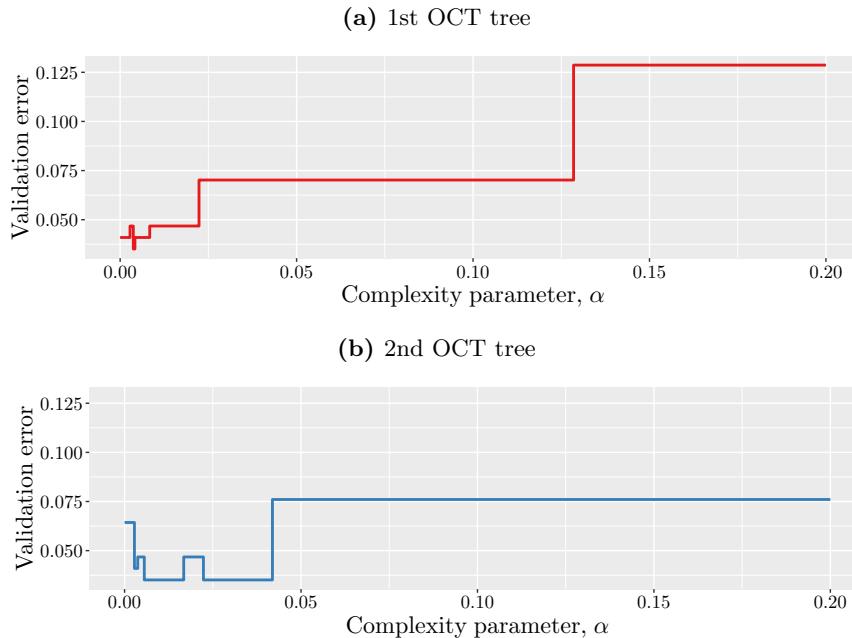
We might simply use the same cost-complexity pruning approach to tune the value of α in the OCT model, but we have found empirically that this method does not lead to tuned values that perform well on new data. To demonstrate this, Figure 8.8 shows results of applying cost-complexity pruning to OCT trees on the same “Breast Cancer Wisconsin” dataset from Figure 8.7. 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 8.8a and 8.8b, 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 8.8b, 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

Figure 8.8: 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.



granularity when selecting the 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 [56], 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* [55], *Arcing* [57] (now known as boosting), and *Random Forests* [58]. 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 [56] 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 8.4, and construct the curve of validation error as a function of complexity parameter:

$$f_j(\alpha) = \sum_{i=1}^{|\alpha^j|} (v_i^j - v_{i-1}^j) \mathbb{1}\{\alpha \geq \alpha_i^j\}, \quad (8.29)$$

where α^j and v^j are the results of the PRUNE function on the j th 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:

$$f(\alpha) = \frac{1}{k} \sum_{j=1}^k f_j(\alpha). \quad (8.30)$$

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 8.6.

We will now illustrate the advantages of this algorithm on the same example as before. Figure 8.9 shows the batch pruning curve for the same “Breast Cancer Wisconsin” dataset as in Figure 8.8. We immediately see that the curve is a lot smoother and has a more obvious minimizing value. To make this comparison clearer, Figure 8.10 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

Figure 8.9: Tuning results using batch pruning on “Breast Cancer Wisconsin” dataset.

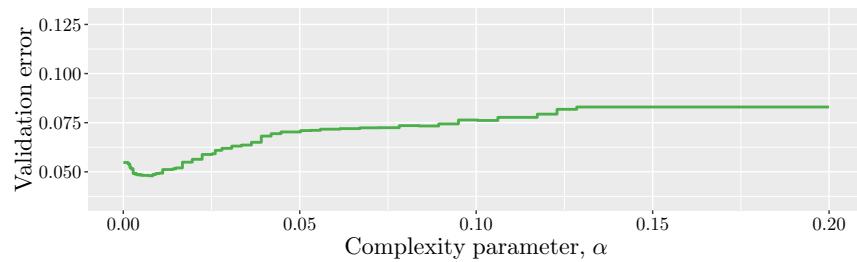
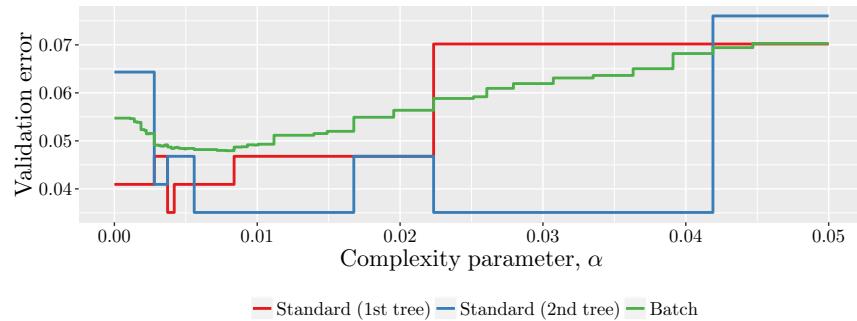


Figure 8.10: Comparison of pruning methods on “Breast Cancer Wisconsin” dataset in the region where validation error is minimized.



Algorithm 8.6 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: $\alpha^j, v^j \leftarrow \text{PRUNE}(trees_j)$
- 4: $f_j(\alpha) \leftarrow \sum_{i=1}^{|\alpha^j|} (v_i^j - v_{i-1}^j) \mathbb{1}\{\alpha \geq \alpha_i^j\}$
- 5: $f(\alpha) \leftarrow \frac{1}{k} \sum_{j=1}^k f_j(\alpha)$
- 6: $v_{\text{best}} \leftarrow \min_{\alpha} f(\alpha)$
- 7: $\mathcal{A} \leftarrow \{\hat{\alpha} : \hat{\alpha} \in \arg \min_{\alpha} f(\alpha)\}$ \triangleright All function minimizers
- 8: $\alpha_{\text{best}} \leftarrow \frac{1}{2}(\min_{\alpha \in \mathcal{A}} \alpha + \max_{\alpha \in \mathcal{A}} \alpha)$ \triangleright Midpoint of minimizers
- 9: **return** $v_{\text{best}}, \alpha_{\text{best}}$

Algorithm 8.7 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** $D = 1, \dots, D_{\text{max}}$ **do**
- 3: $\mathbb{T} \leftarrow$ decision tree trained with maximum depth D
- 4: $\alpha, v \leftarrow \text{BATCHTUNECP}(\mathbb{T}, k)$
- 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** $\alpha_{\text{best}}, D_{\text{best}}$

the power of the batch 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 8.7.

In Algorithm 8.7, 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 $N_{\text{min}} = 1$, and so it

is unlikely that tuning this parameter will be worth the additional cost. However, it is trivial to modify Algorithm 8.7 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.

8.5 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 [200]. 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 8.7.

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 [200] 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.

comparison with ctree and if so where?

Implementation Details

The Optimal Classification Tree method as detailed in Section 8.3 was implemented in the JULIA programming language, which is a high-level, high-performance dynamic programming language for technical computing [44]. We used Algorithm 8.7 to tune the hyperparameters.

For testing the performance of CART, we used the RPART package [250] in the R programming language [224]. 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 8.4.

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 8.11 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 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

Figure 8.11: Synthetic experiments showing the effect of training set size.

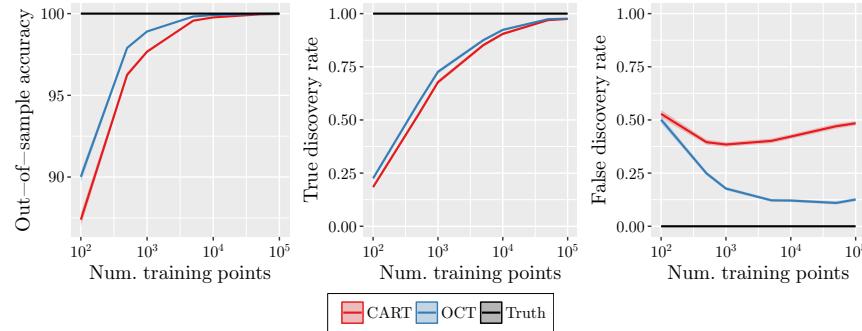
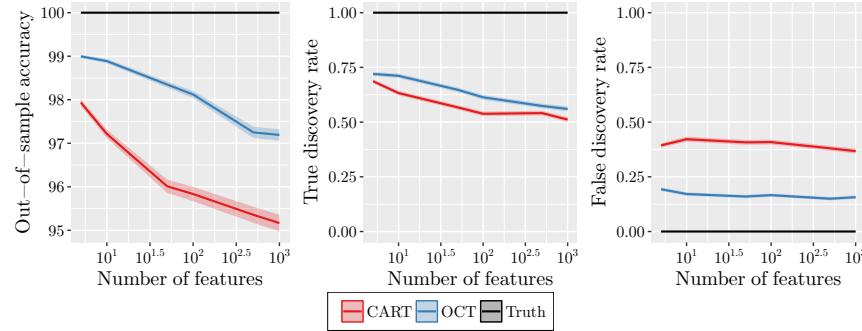
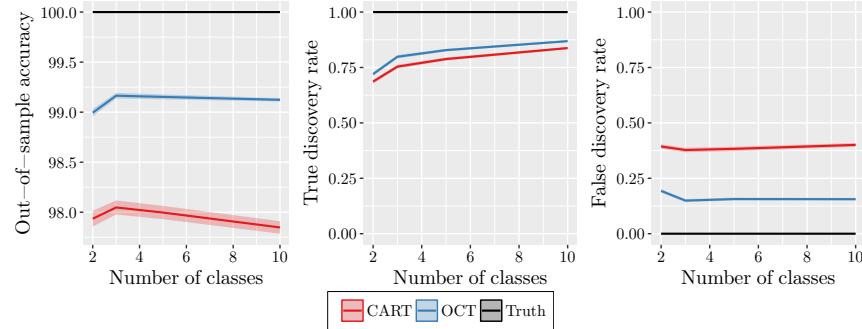


Figure 8.12: Synthetic experiments showing the effect of number of features.



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.

Figure 8.13: Synthetic experiments showing the effect of number of classes.



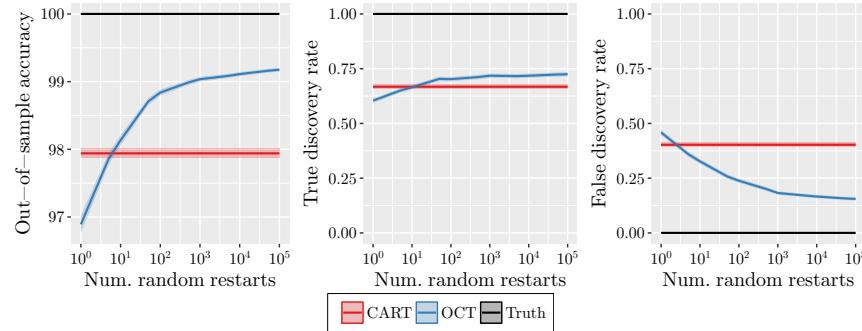
The effect of dimensionality

The second experiment considers the effect of the dimensionality of the problem, and the results are shown in Figure 8.12. 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 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 8.13 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 8.14: 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 8.14. 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 parameter of the experiment. Figure 8.15 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

Figure 8.15: Synthetic experiments showing the effect of feature noise.

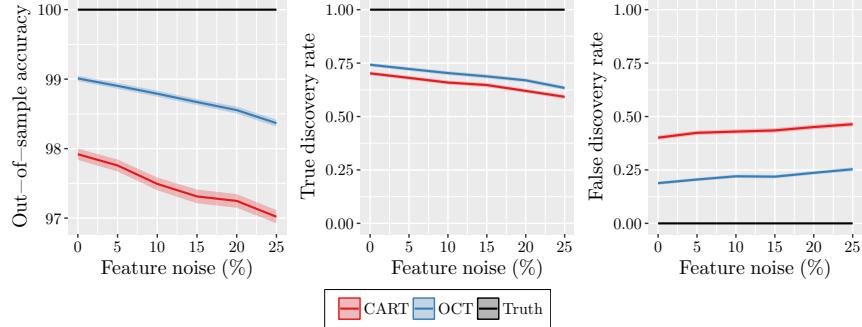
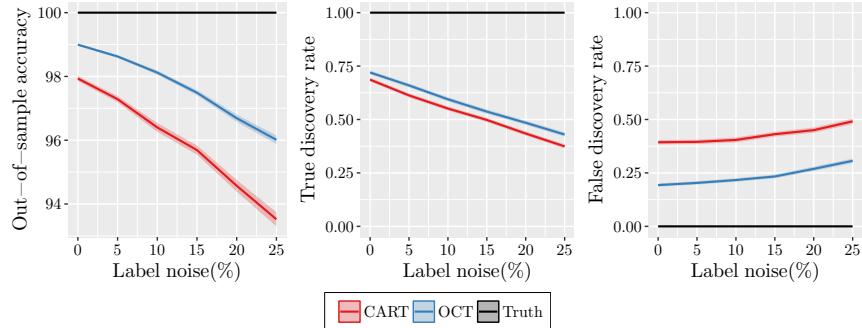


Figure 8.16: Synthetic experiments showing the effect of label noise.



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 8.16 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 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.

8.6 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 [172]. 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 8.5. We trained Optimal Classification Trees of maximum depths from 1–10 on all datasets with $R = 500$ random restarts, using Algorithm 8.7 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.

A discussion on running time and hardware?

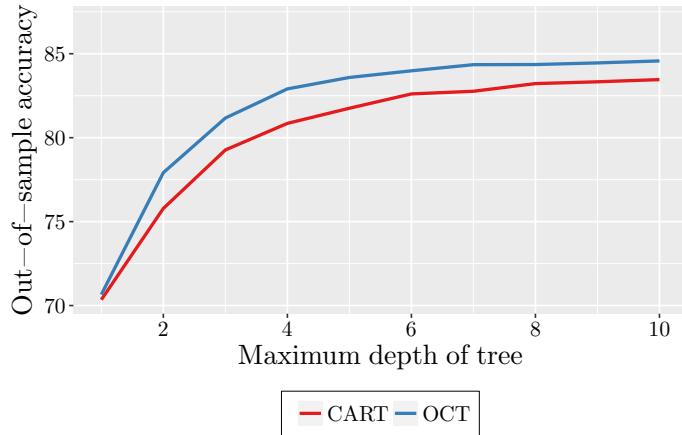
Optimal Classification Trees vs. CART

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.

Table 8.1: 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.

Name	Dataset			Mean out-of-sample accuracy		Mean improvement
	n	p	K	CART	OCT	
acute-inflammations-1	120	6	2	95.3	100.0	+4.67 ± 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 ± 1.08
banknote-authentication	1372	4	2	98.0	98.9	+0.87 ± 0.36
blood-transfusion	748	4	2	77.2	78.1	+0.86 ± 0.80
breast-cancer-diagnostic	569	30	2	90.9	92.7	+1.82 ± 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 ± 0.47
car-evaluation	1728	15	4	91.3	92.4	+1.16 ± 1.18
chess-king-rook-vs-king-pawn	3196	37	2	98.9	99.3	+0.43 ± 0.11
climate-model-crashes	540	18	2	91.3	92.3	+1.04 ± 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 ± 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 ± 2.59
dermatology	358	34	6	95.1	95.3	+0.22 ± 0.90
echocardiogram	61	6	2	72.0	73.3	+1.33 ± 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 ± 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 ± 1.38
hill-valley	606	100	2	49.8	52.6	+2.78 ± 0.68
image-segmentation	210	19	7	77.7	86.8	+9.06 ± 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 ± 3.12
iris	150	4	3	93.5	94.6	+1.08 ± 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 ± 6.77
monks-problems-2	169	11	2	59.5	60.0	+0.47 ± 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 ± 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 ± 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 ± 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 ± 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 ± 0.48
skin-segmentation	245057	3	2	99.7	99.9	+0.17 ± 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 ± 0.34
spect-heart	80	22	2	61.0	62.0	+1.00 ± 4.00
spectf-heart	80	44	2	72.0	76.0	+4.00 ± 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 ± 0.57
teaching-assistant	151	52	3	56.8	63.2	+6.49 ± 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 ± 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 ± 2.47

Figure 8.17: Mean out-of-sample accuracy for each method across all 60 datasets.



The entire set of mean out-of-sample accuracies on each dataset for both methods at depth 10 is provided in Table 8.1, which reports the size and dimension of the dataset, the number of class labels K , and the average out-of-sample accuracy of each 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 8.17 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 8.2, 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 the same accuracy as CART with much fewer splits. This indicates that OCT is able to better discover

Table 8.2: 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 ± 0.16	0.0481
2	75.78	77.92	+2.13 ± 0.32	~10 ⁻¹⁰
3	79.27	81.17	+1.90 ± 0.38	~10 ⁻⁶
4	80.85	82.91	+2.05 ± 0.37	~10 ⁻⁷
5	81.75	83.59	+1.83 ± 0.38	~10 ⁻⁶
6	82.61	83.98	+1.38 ± 0.31	~10 ⁻⁵
7	82.76	84.35	+1.58 ± 0.30	~10 ⁻⁷
8	83.22	84.36	+1.14 ± 0.28	~10 ⁻⁴
9	83.33	84.45	+1.12 ± 0.27	~10 ⁻⁴
10	83.46	84.57	+1.11 ± 0.27	~10 ⁻⁴

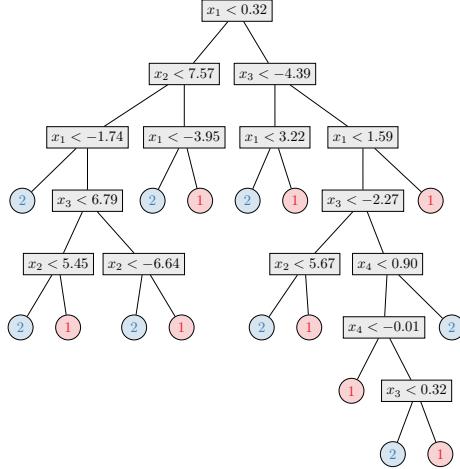
the truth in the data and give trees with truly meaningful splits.

Figure 8.18 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 8.18a, 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 8.18b, 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 8.18c, 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.

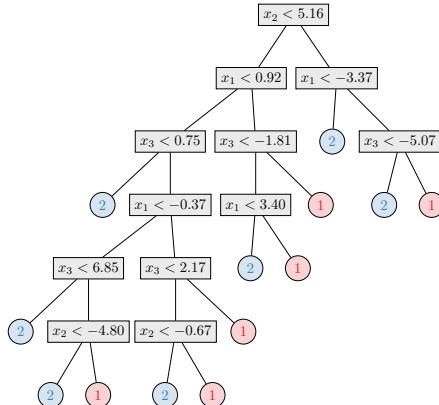
Table 8.3 shows the number of times CART and OCT were the strongest method for a dataset across the sample of 60 datasets, broken down according to the maximum depth of the trees. We see that OCT performs strongest on significantly more datasets than CART. At depths 1 and 2, OCT wins about three times 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),

Figure 8.18: 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)

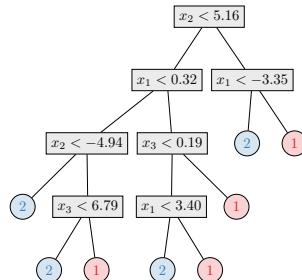


Table 8.3: Number of datasets where CART and OCT were strongest across all 60 datasets.

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

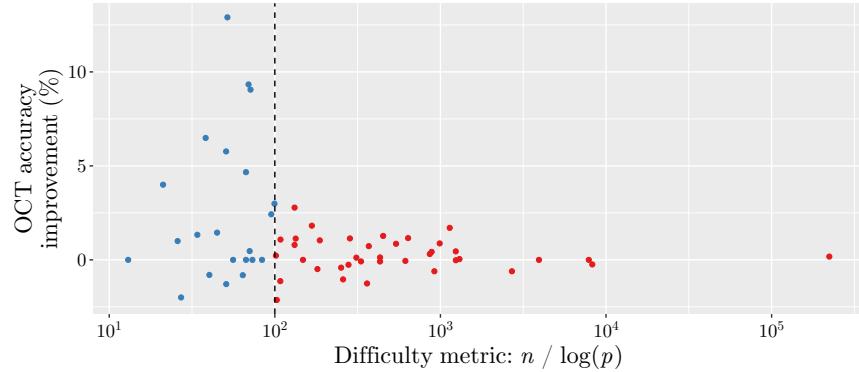
yet it still performs strongest in about twice as many datasets as CART. These results show that OCT is able to outperform CART in a significant majority of datasets, in addition to having a higher mean out-of-sample accuracy across the sample of datasets.

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 8.5, 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 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 8.19 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

Figure 8.19: Improvement in out-of-sample accuracy for OCT over CART for each of the 60 datasets, according to ratio of n and $\log(p)$.



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 8.5 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.

8.7 Concluding Remarks

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.

8.8 Notes and Sources

Dimitris Bertsimas and Jack Dunn proposed optimal classification trees [26]. The treatment here is from Jack Dunn’s PhD thesis [85].

Chapter 9

Optimal Classification Trees with Hyperplane Splits

In a choice between accuracy and interpretability, doctors will go for interpretability.

– Leo Breiman;

Contents

- 9.1. OCT with Hyperplanes via MIO
- 9.2. OCT with Hyperplanes via Local Search
- 9.3. Interpretability of Parallel and Hyperplane Trees
- 9.4. Experiments with Synthetic Datasets
- 9.5. Experiments with Real-World Datasets
- 9.6. Concluding Remarks
- 9.7. Notes and Sources

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 [133].

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) [54], 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) [132] which uses simulated annealing to perturb an existing hyperplane towards optimality. These perturbation approaches were combined to give OC1 [201], 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 [259], support vector machines [16], linear discriminant analysis [180, 181], and Householder transformations [271].

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 8 to generate trees with hyperplane splits, and show that these trees significantly improve upon state-of-the-art methods.

9.1 OCT with Hyperplanes via MIO

In Section 8.2, 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 (8.4) and instead choose $\mathbf{a}_t \in [-1, 1]^p$ at each branch node t . We must modify (8.2) to account for 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 $|a_{jt}|$:

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

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 (8.3) with:

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

Now we consider the split constraints (8.9) and (8.14). 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:

$$\mathbf{a}_m^T \mathbf{x}_i < b_m + 2(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \quad (9.1)$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - 2(1 - z_{it}), \quad \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \quad (9.2)$$

Finally, as before we need to convert the strict inequality in (9.1) 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 j th feature is used in the t th 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:

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

$$\sum_{j=1}^p s_{jt} \geq d_t, \quad \forall t \in \mathcal{T}_B.$$

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

are these actually needed?

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:

$$\begin{aligned} \min \quad & \frac{1}{\hat{L}} \sum_{t \in \mathcal{T}_L} L_t + \alpha \cdot C && (9.3) \\ \text{s.t.} \quad & L_t \geq N_t - N_{kt} - n(1 - c_{kt}), && \forall k \in [K], t \in \mathcal{T}_L, \\ & L_t \leq N_t - N_{kt} + nc_{kt}, && \forall k \in [K], t \in \mathcal{T}_L, \\ & L_t \geq 0, && \forall t \in \mathcal{T}_L, \\ & N_{kt} = \sum_{i: y_i=k} z_{it}, && \forall k \in [K], t \in \mathcal{T}_L, \\ & N_t = \sum_{i=1}^n z_{it}, && \forall t \in \mathcal{T}_L, \\ & \sum_{k=1}^K c_{kt} = l_t, && \forall t \in \mathcal{T}_L, \\ & C = \sum_{t \in \mathcal{T}_B} \sum_{j=1}^p s_{jt}, && \\ & \mathbf{a}_m^T \mathbf{x}_i + \mu \leq b_m + (2 + \mu)(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \\ & \mathbf{a}_m^T \mathbf{x}_i \geq b_m - 2(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \\ & \sum_{t \in \mathcal{T}_L} z_{it} = 1, && \forall i \in [n], \\ & z_{it} \leq l_t, && \forall t \in \mathcal{T}_L, \\ & \sum_{i=1}^n z_{it} \geq N_{\min} l_t, && \forall t \in \mathcal{T}_L, \\ & \sum_{j=1}^p \hat{a}_{jt} \leq d_t, && \forall t \in \mathcal{T}_B, \\ & \hat{a}_{jt} \geq a_{jt}, && \forall t \in \mathcal{T}_B, j \in [p], \\ & \hat{a}_{jt} \geq -a_{jt}, && \forall t \in \mathcal{T}_B, j \in [p], \\ & -s_{jt} \leq a_{jt} \leq s_{jt}, && \forall t \in \mathcal{T}_B, j \in [p], \\ & s_{jt} \leq d_t, && \forall t \in \mathcal{T}_B, j \in [p], \\ & \sum_{j=1}^p s_{jt} \geq d_t, && \forall t \in \mathcal{T}_B, \\ & -d_t \leq b_t \leq d_t, && \forall t \in \mathcal{T}_B, \end{aligned}$$

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

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.

9.2 OCT with Hyperplanes via Local Search

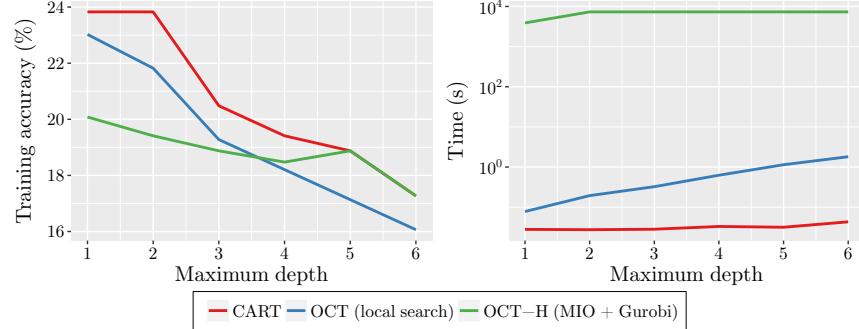
In this section, we extend the local search heuristic presented in Section 8.3 to generate trees with hyperplane splits, thereby avoiding the scaling issues of MIO-based approaches and allowing us to generate Optimal Classification Trees with Hyperplanes in a tractable and effective manner.

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 fast with the depth of the tree and number of training points, quickly causing the MIO problem to become intractable. Figure 9.1 shows the performance of this MIO formulation for OCT-H on the “Blood Transfusion Service Center” dataset from the UCI Machine Learning Repository [172], 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 8.3 but with modifications to allow generation trees with hyperplane splits. To do this, we simply augment the procedure `OPTIMIZENODEPARALLEL` from Algorithm 8.2 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 [54] and OC1 [201]. Underpinning this approach is the observation that it is possible to optimize

Figure 9.1: Training accuracy (%) and running time for each method on “Blood Transfusion Service Center” dataset.



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

$$\hat{V}_i \triangleq \sum_{j=1}^p \hat{a}_j x_{ij} - \hat{b} < 0, \quad (9.4)$$

otherwise the upper branch will be taken.

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

$$\hat{V}_i + (a_k - \hat{a}_k)x_{ik} < 0. \quad (9.5)$$

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

$$a_k < \frac{\hat{a}_k x_{ik} - \hat{V}_i}{x_{ik}}, \quad (9.6)$$

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

$$U_{ik} = \begin{cases} \frac{\hat{a}_k x_{ik} - \hat{V}_i}{x_{ik}}, & x_{ik} > 0, \\ -\infty, & x_{ik} = 0, V_i \geq 0, \\ +\infty, & x_{ik} = 0, V_i < 0. \end{cases} \quad (9.7)$$

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 8.3, 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 k th coefficient has been deleted:

$$W_{ik} \triangleq \hat{V}_i + \hat{b} - \hat{a}_k x_{ik}, \quad (9.8)$$

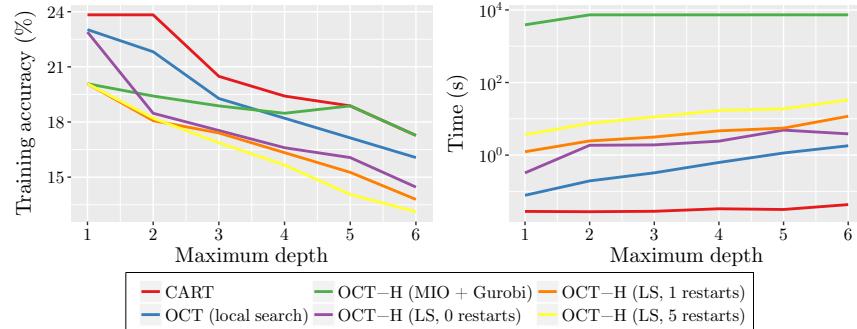
and therefore point i follows the lower branch of the split with the k th coefficient deleted if $W_{ik} < b$. We can then sort these critical values and scan them in the same way as above to find the optimal value for b that minimizes the loss function. This gives us the optimal threshold for the hyperplane after the k th 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 9.1.

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. This process is detailed in Algorithm 9.2, and the full local search procedure for trees with hyperplane splits is then simply the same as shown in Algorithm 8.1 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 9.2 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 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

Figure 9.2: Training accuracy (%) and running time for each method on “Blood Transfusion Service Center” dataset.



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, 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 8.3 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 9.1, which is very similar to the `BESTPARALLELSPPLIT` in Algorithm 8.3. 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 `BESTPARALLELSPPLIT`, 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 9.2, which is again nearly identical to its parallel counterpart OPTIMIZENODEPARALLEL in Algorithm 8.2, 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 8.1 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 $(K + \log n)$.

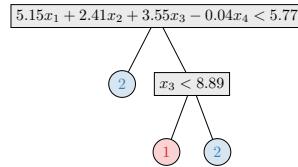
9.3 Interpretability of Parallel and Hyperplane Trees

In this section, we discuss the tradeoff between the interpretability and accuracy of decision trees with parallel and hyperplane splits to evaluate the impact of allowing for hyperplane splits on the interpretability of the tree.

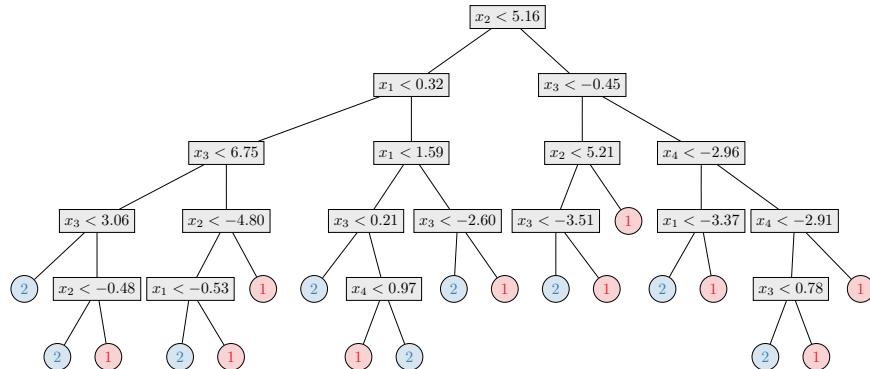
If we consider a single split in a decision tree, there is no question that a parallel split is more easily interpreted than a hyperplane split, particularly as the number of features used in the hyperplane split increases.

Figure 9.3: 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%.

(a) Optimal tree with hyperplane splits (depth 2)



(b) Optimal tree with parallel splits (depth 5)



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.

An example of such a scenario is shown in Figure 9.3, which shows two trees that were trained on the “Banknote Authentication” dataset from the UCI Machine Learning Repository [172], 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 9.3 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.

9.4 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.

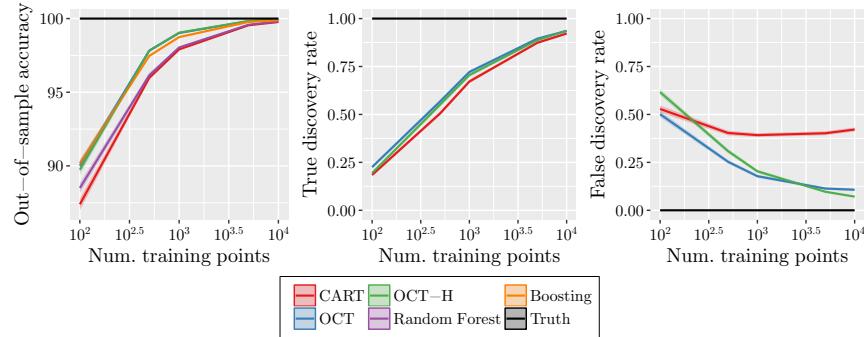
The experimental setup is the same as in Section 8.5. We randomly generate decision trees and then use these trees to generate training and testing datasets. Unless otherwise mentioned, we use the same default parameters as in Section 8.5 and use $H = 5$ random hyperplane restarts in the OCT-H method.

In each experiment, we compare the performance of CART, OCT and OCT-H, along with random forests (using the `RANDOMFOREST` package in R [170]) and gradient-boosted trees (using the `XGBOOST` library [72] 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 8.5, except with the addition of the three new methods. There are two main

Figure 9.4: Synthetic experiments showing the effect of training set size.



aims in repeating these tests 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 9.4. 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 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 9.5. We can see that OCT, OCT-H and boosting all have roughly the same out-

Figure 9.5: Synthetic experiments showing the effect of number of features.

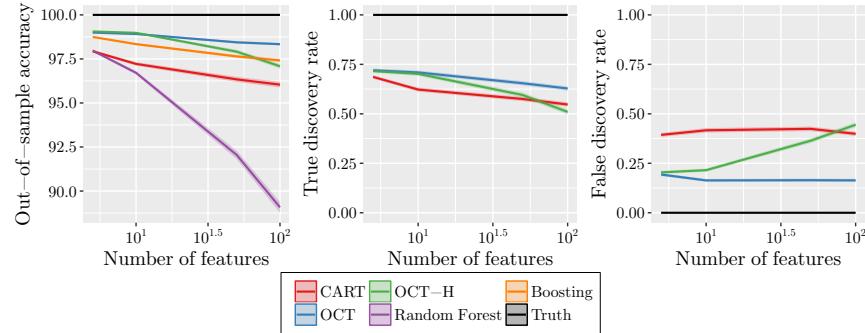
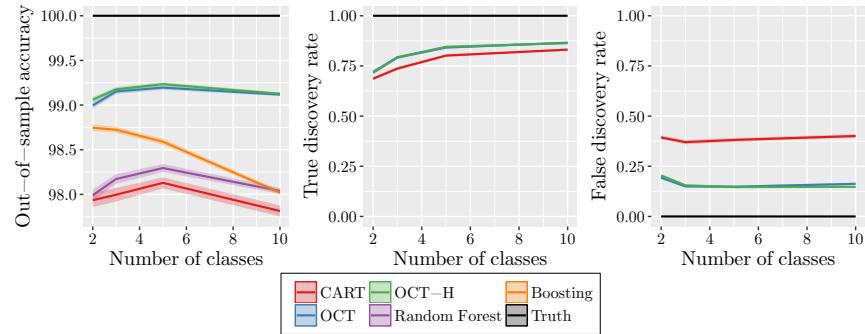


Figure 9.6: Synthetic experiments showing the effect of number of classes.

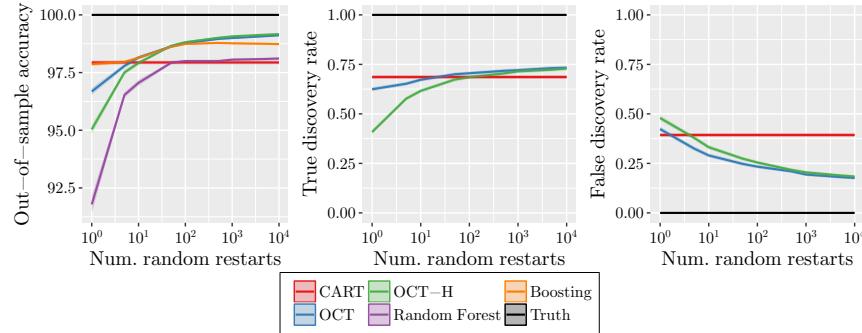


of-sample accuracy at all dimensions, while the accuracy of random forests quickly falls as the dimension increases. The TDR of OCT-H is initially similar to OCT, and then falls to the same level as CART at the higher numbers of features. Similarly, the FDR starts close to OCT, then quickly rises past 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 9.6. 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

Figure 9.7: Synthetic experiments showing the effect of number of random restarts.



number of classes is increased, while the accuracy 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 9.7 shows the results. 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 8.5, we added uniform noise to $f\%$ of the training points. The results are shown in Figure 9.8. We can see that all methods decline in accuracy as the

Figure 9.8: Synthetic experiments showing the effect of feature noise.

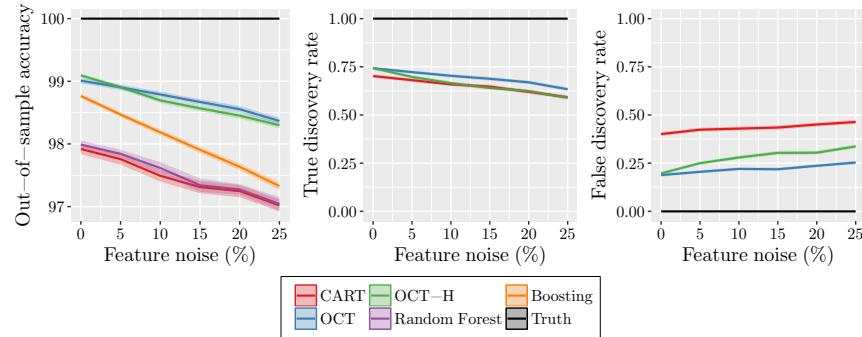
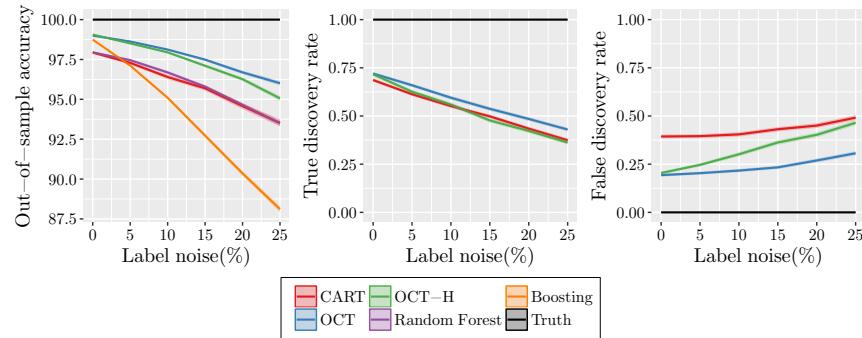


Figure 9.9: Synthetic experiments showing the effect of label noise.



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 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 9.9 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

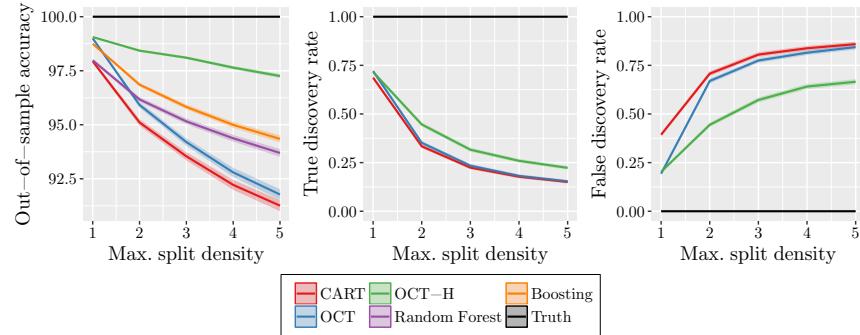
We now summarize the results of these repeated experiments with parallel ground truth trees from Section 8.5. In all cases, OCT and OCT-H had near-identical results, indicating that the additional power of OCT-H is not leading it to overfit with more complicated trees, and instead it is learning the 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

Figure 9.10: Synthetic experiments showing the effect of split density.

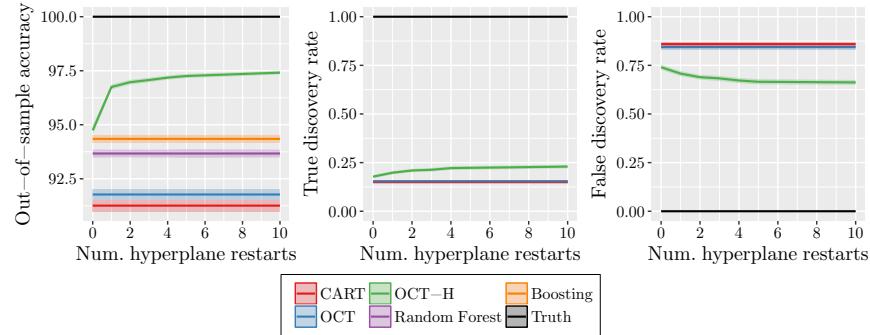


OCT-H maintains sparsity and only uses as many features as are required by each split, or whether it simply overfits with very dense splits.

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

Figure 9.11: Synthetic experiments showing the effect of number of random hyperplane restarts.



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 9.11 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.

9.5 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 8.6. 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.

Experimental Setup

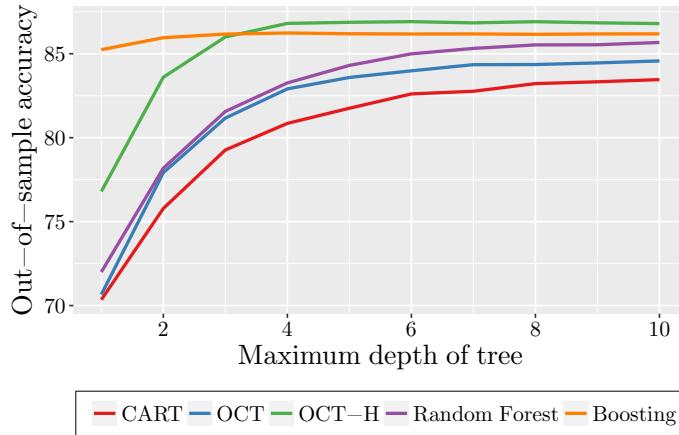
We used the same experimental setup and sample of 60 datasets as in Section 8.6 for these experiments. In addition to running CART and OCT as described in Section 8.6, we ran OCT-H with $R = 100$ random restarts and $H = 10$ hyperplane restarts, using Algorithm 8.4 to tune the values of the complexity 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 9.4.

Results and Discussion

Figure 9.12 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 9.3 that trees with hyperplane splits can achieve comparable (or indeed better) accuracies to

Figure 9.12: Mean out-of-sample accuracy for each method across all 60 datasets.



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 9.1 shows both the accuracy and performance rank of each method on each of the datasets. These ranks allow 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

Table 9.1: 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 9.2: 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

which have statistically significant differences in performance. Table 9.2 shows the results of these comparisons. We follow the approach recommended by [82] and [108] 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 then adjusted using the Holm-Bonferroni method [139] to control the familywise error rate. The comparisons are presented in order of significance. We see that CART is statistically 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.

9.6 Concluding Remarks

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.

9.7 Notes and Sources

Dimitris Bertsimas and Jack Dunn proposed optimal classification trees with hyperplanes [26]. The treatment here is from Jack Dunn's PhD thesis [85].

Algorithm 9.1 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  $k = \text{shuffle}(1, \dots, p)$  do            $\triangleright \text{loop over all dimensions}$ 
4:      $\text{values} \leftarrow \{U_{ik} : i = 1, \dots, n\}$      $\triangleright \text{critical values for perturbing}$ 
5:     sort values in ascending order
6:     for  $i = 1, \dots, n - 1$  do            $\triangleright \text{loop over all split placements}$ 
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  $\text{minleafsize}(\mathbb{T}) \geq N_{\min}$  then            $\triangleright \text{check feasibility}$ 
10:         $\text{error} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$ 
11:        if  $\text{error} < \text{error}_{\text{best}}$  then            $\triangleright \text{save split if better}$ 
12:           $\text{error}_{\text{best}} \leftarrow \text{error}$ 
13:           $\mathbb{T}_{\text{best}} \leftarrow \mathbb{T}$ 
14:      if  $a_k \neq 0$  at root of  $\mathbb{T}$  then            $\triangleright \text{can only delete if present}$ 
15:         $\text{values} \leftarrow \{W_{ik} : i = 1, \dots, n\}$      $\triangleright \text{critical values for deleting}$ 
16:        sort values in ascending order
17:        for  $i = 1, \dots, n - 1$  do            $\triangleright \text{loop over all split placements}$ 
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  $\text{minleafsize}(\mathbb{T}) \geq N_{\min}$  then            $\triangleright \text{check feasibility}$ 
22:             $\text{error} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$ 
23:            if  $\text{error} < \text{error}_{\text{best}}$  then            $\triangleright \text{save split if better}$ 
24:               $\text{error}_{\text{best}} \leftarrow \text{error}$ 
25:               $\mathbb{T}_{\text{best}} \leftarrow \mathbb{T}$ 
26: until  $\text{error}_{\text{prev}} = \text{error}_{\text{best}}$             $\triangleright \text{no further improvement possible}$ 
27: return  $\mathbb{T}, \text{error}_{\text{best}}$ 

```

Algorithm 9.2 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:    $\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}} \leftarrow \text{new leaf nodes}$ 
5:  $\text{error}_{\text{best}} \leftarrow \text{loss}(\mathbb{T}, \mathbf{X}, \mathbf{y})$                                 ▷ error of current root
6:
7:  $\mathbb{T}_{\text{para}}, \text{error}_{\text{para}} \leftarrow \text{BESTPARALLELSPLIT}(\mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}, \mathbf{X}, \mathbf{y})$ 
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  $h = 1, \dots, H + 1$  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:      $\mathbb{T}_{\text{start}} \leftarrow \text{branch node } \mathbf{a}^T \mathbf{x} < b \text{ with children } \mathbb{T}_{\text{lower}}, \mathbb{T}_{\text{upper}}$ 
23:      $\mathbb{T}_{\text{hyper}}, \text{error}_{\text{hyper}} \leftarrow \text{BESTHYPERPLANESPLIT}(\mathbb{T}_{\text{start}}, \mathbf{X}, \mathbf{y})$ 
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  $\mathbb{T}$ 
```

Chapter 10

Optimal Regression Trees with Constant Predictions

Make things as simple as possible, but not simpler.

– Albert Einstein

Contents

- 10.1. Review of Regression Tree Methods
- 10.2. ORT with Constant Predictions via MIO
- 10.3. ORT with Constant Predictions via Local Search
- 10.4. Experiments with Synthetic Datasets
- 10.5. Experiments with Real-World Datasets
- 10.6. Concluding Remarks
- 10.7. Notes and Sources

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 interpretability. 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 [54, 160, 143]. 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.

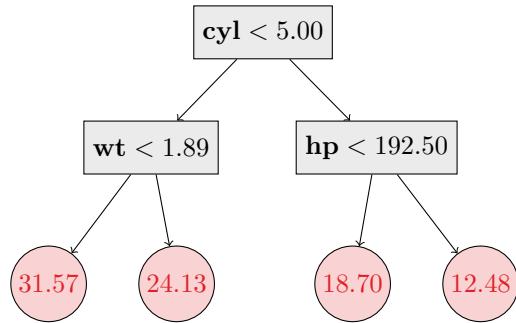
In this chapter, we apply the Optimal Trees methodology for classification problems from Chapters 8 and 9 to the regression problem, yielding a procedure for generating Optimal Regression Trees with constant predictions in each leaf. Our goal is to investigate the effects of solving the regression tree 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 11.

10.1 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

Figure 10.1: 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.



classification.

Figure 10.1 shows an example of a regression tree trained on the “mtcars” dataset [136]. 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 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 (8.1) 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.

10.2 ORT with Constant Predictions via MIO

In this section, we follow the example of Optimal Classification Trees from Chapters 8 and 9 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:

$$f_i = \sum_{t \in \mathcal{T}_L} \beta_{0t} z_{it}, \quad \forall i \in [n]. \quad (10.1)$$

Since only one of the $z_{it} = 1$, indicating that the t th 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:

$$-M_f(1 - z_{ik}) \leq f_i - \beta_{0t} \leq M_f(1 - z_{ik}), \quad \forall i \in [n], t \in \mathcal{T}_L, \quad (10.2)$$

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 i th point as follows:

$$L_i = |f_i - y_i|, \quad \forall i \in [n], \quad (10.3)$$

which can be linearized to give

$$L_i \geq +f_i - y_i, \quad \forall i \in [n], \quad (10.4)$$

$$L_i \geq -f_i + y_i, \quad \forall i \in [n], \quad (10.5)$$

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:

$$L_i \geq (f_i - y_i)^2, \quad \forall i \in [n], \quad (10.6)$$

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:

$$\min \frac{1}{\hat{L}} \sum_{i=1}^n L_i + \alpha \cdot C, \quad (10.7)$$

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 (8.24) for parallel splits or in Problem (9.3) for hyperplane splits. For instance, the following is the complete *Optimal Regression Trees* (ORT) model with squared loss:

$$\begin{aligned} \min \quad & \frac{1}{\hat{L}} \sum_{i=1}^n L_i + \alpha \cdot C && (10.8) \\ \text{s.t.} \quad & L_i \geq (f_i - y_i)^2, && \forall i \in [n], \\ & f_i - \beta_{0t} \geq -M_f(1 - z_{ik}), && \forall i \in [n], t \in \mathcal{T}_L, \\ & f_i - \beta_{0t} \leq +M_f(1 - z_{ik}), && \forall i \in [n], t \in \mathcal{T}_L, \\ & C = \sum_{t \in \mathcal{T}_B} d_t, && \\ & \mathbf{a}_m^\top \mathbf{x}_i \geq b_m - (1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{R}(t), \\ & \mathbf{a}_m^\top (\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), && \forall i \in [n], t \in \mathcal{T}_L, m \in \mathcal{L}(t), \\ & \sum_{t \in \mathcal{T}_L} z_{it} = 1, && \forall i \in [n], \\ & z_{it} \leq l_t, && \forall t \in \mathcal{T}_L, \\ & \sum_{i=1}^n z_{it} \geq N_{\min} l_t, && \forall t \in \mathcal{T}_L, \\ & \sum_{j=1}^p a_{jt} = d_t, && \forall t \in \mathcal{T}_B, \end{aligned}$$

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

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 than 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.

10.3 ORT with Constant Predictions via Local Search

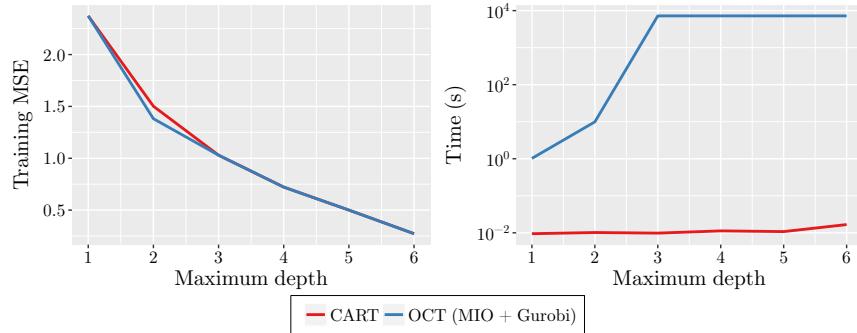
In this section, we modify the local search heuristic developed in Sections 8.3 and 9.2 to generate regression trees, allowing us to generate Optimal Regression Trees both without (ORT) and with (ORT-H) hyperplane splits.

The MIO formulations for the ORT and ORT-H problems in Section 10.2 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 10.2 demonstrates the scaling issues of the MIO approach. We trained CART and ORT trees on the “Hybrid Vehicle Prices” dataset made available in [173], 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

Figure 10.2: Training mean squared error and running time for each method on “Hybrid Vehicle Prices” dataset.



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 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 8.1 and changing the definition of the loss function $L(\mathbb{T}, \mathbf{X}, \mathbf{y})$ in (8.26) 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 8.1 described in Section 9.2.

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 8.3 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.

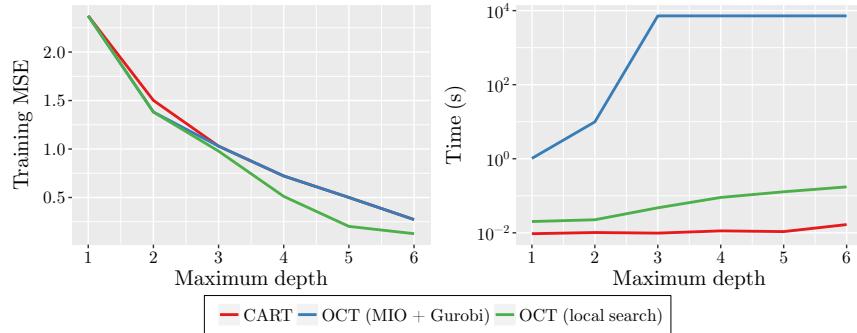
Next, we consider the cost of ORT and ORT-H with absolute loss. This requires us to calculate the median of the labels at each leaf followed by the absolute loss. There are approaches for updating the median and absolute loss that run in $O(\log n)$ time [257]. This means that the cost of BESTPARALLELSPLIT 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 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 10.3 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.

Figure 10.3: Training mean squared error and running time for each method on “Hybrid Vehicle Prices” dataset.



This gives evidence that the solutions found by the local search procedure can indeed be optimal, albeit without a certificate of optimality.

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

The experimental setup mirrors that for classification in Sections 8.5 and 9.4 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.

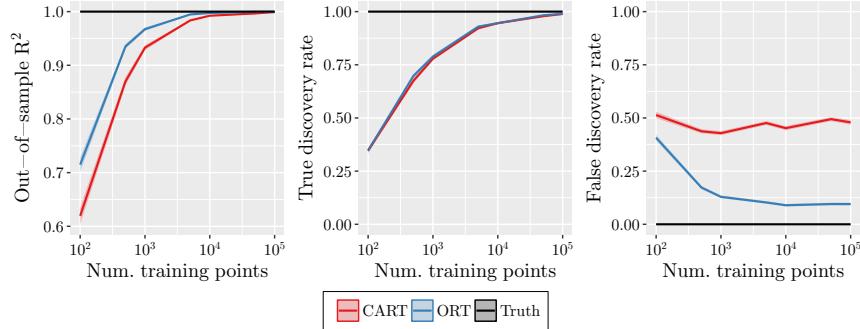
We trained ORT and ORT-H with the squared loss due to the additional computational overhead of the local search using absolute loss.

Each method was tuned to maximize R^2 using the same procedures as Sections 8.5 and 9.4. 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 ORT. 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 10.4: Synthetic experiments showing the effect of training set size.



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 10.4. 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 8.5, 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.

The effect of dimensionality

The second experiment investigates the effect of the dimensionality of the problem, and the results are shown in Figure 10.5. 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

Figure 10.5: Synthetic experiments showing the effect of number of features.

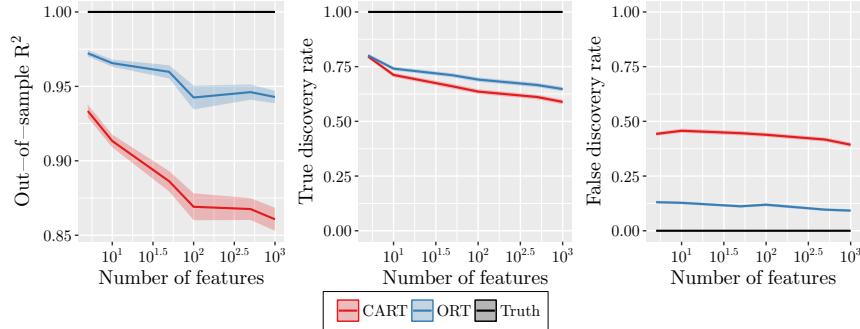
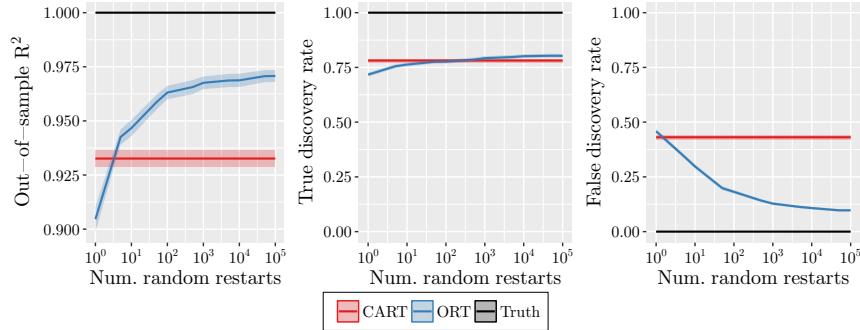


Figure 10.6: Synthetic experiments showing the effect of number of random restarts.



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 10.6. 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 capture most of the the advantage of Optimal Trees. Further

Figure 10.7: Synthetic experiments showing the effect of feature noise.

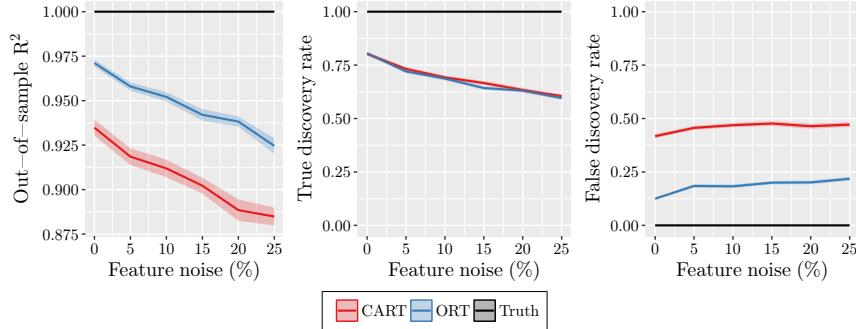
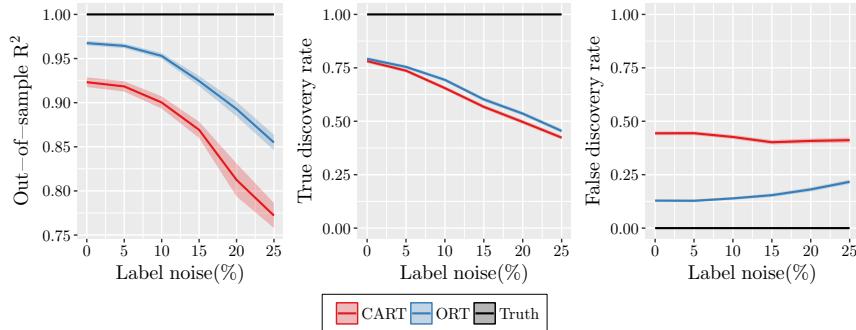


Figure 10.8: Synthetic experiments showing the effect of label noise.



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 10.7 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 8.16 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 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 8.5 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.

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.

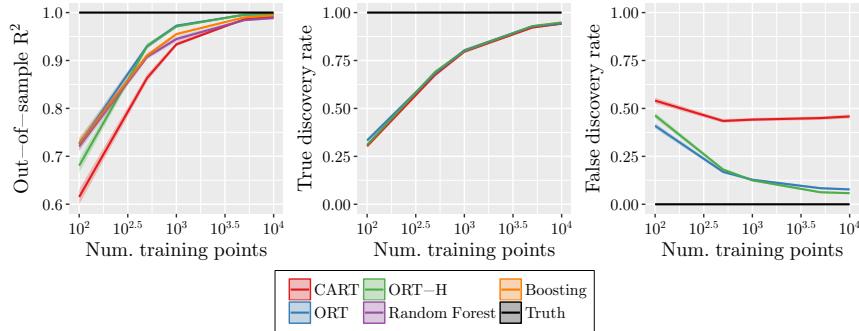
Ground truth trees with parallel splits

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 10.9. We see that

Figure 10.9: Synthetic experiments showing the effect of training set size.



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 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 10.10. We see trends that are generally similar to those for classification in Figure 9.5. 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 OCT. All three tree methods perform similarly in TDR. The FDR of CART and ORT remains constant

Figure 10.10: Synthetic experiments showing the effect of number of features.

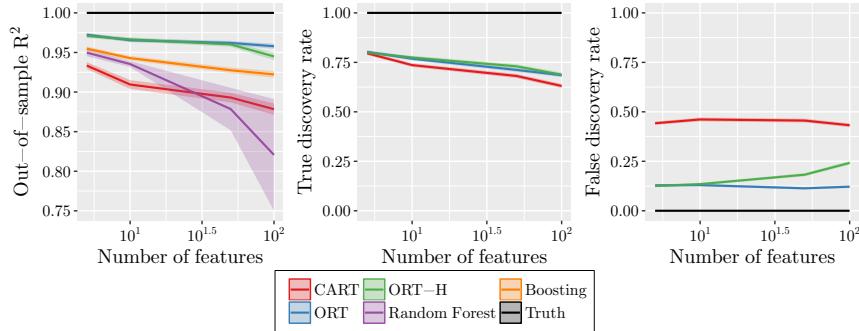
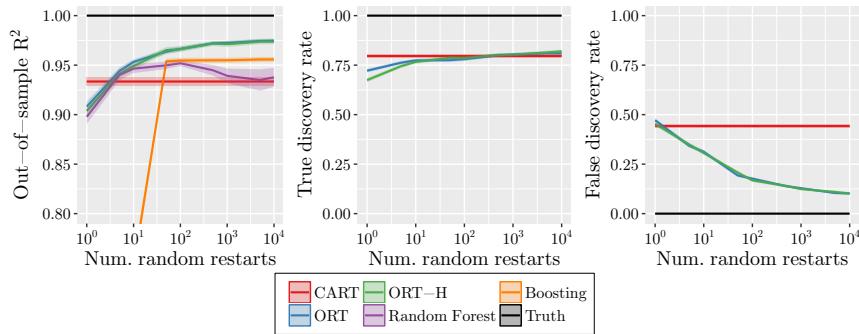


Figure 10.11: Synthetic experiments showing the effect of number of random restarts.

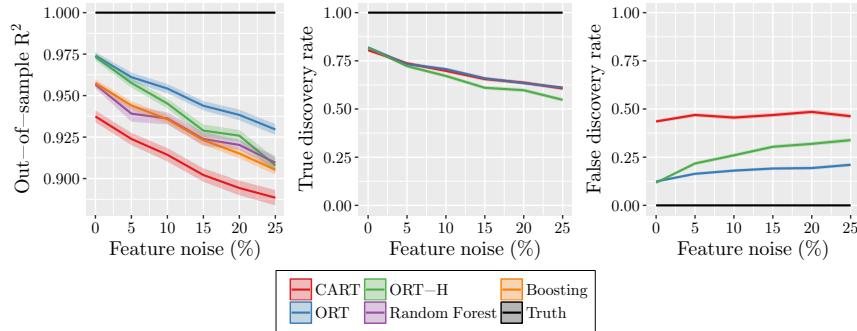


with an increasing number of features, but the FDR for ORT-H begins to worsen. As with the out-of-sample R^2 , this deterioration as the number of features increases is less pronounced than it was for OCT-H compared to OCT.

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 10.11. 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

Figure 10.12: Synthetic experiments showing the effect of feature noise.



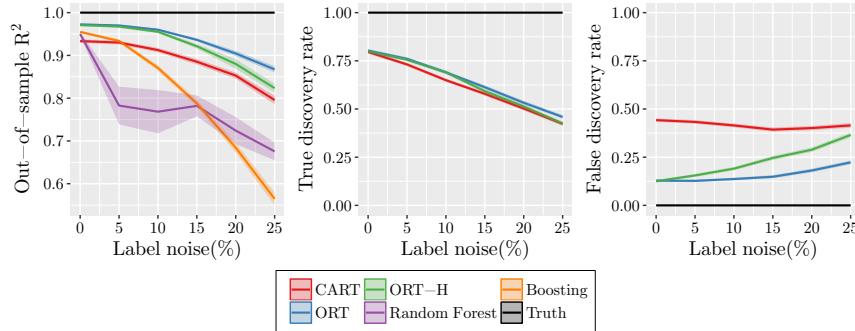
between that of CART and boosting and has much higher variance than the other methods. The TDR and FDR of ORT and ORT-H is very similar, with the TDR leveling out around 0.8, similar to CART, and the FDR decreasing towards zero, compared to around 0.45 for CART.

The effect of noise in the training data

The next experiment considers adding feature noise to the training data, and Figure 10.12 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 10.13. The results for all methods largely mirror those for classification in Figure 9.9, with the exception of random forests, which seem to perform significantly worse under label noise for regression compared to classification. The optimal tree methods and CART all decrease slightly in out-of-sample performance

Figure 10.13: Synthetic experiments showing the effect of label noise.



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

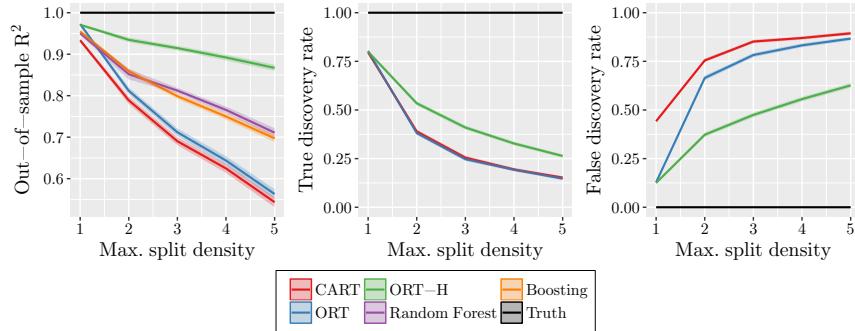
We now summarize the results of these comparisons of all methods on data generated from ground truth trees with parallel splits. In most cases, ORT and ORT-H performed similarly and were the best among all the methods. There were some cases where the performance of ORT-H was reduced compared to ORT, 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.

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.

Figure 10.14: Synthetic experiments showing the effect of split density.



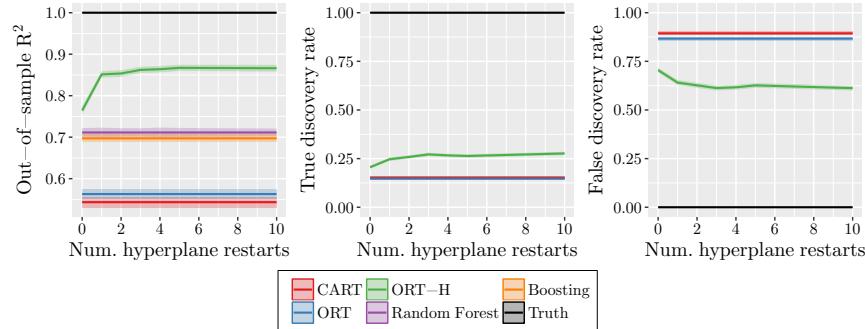
The effect of split density

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 10.14, and closely mirror the corresponding results for classification in Figure 9.10. 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.

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 10.15 again closely match those for the corresponding test for classification in Figure 9.11. 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.

Figure 10.15: Synthetic experiments showing the effect of number of random hyperplane restarts.



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.

10.5 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 8.6 and 9.5. We used a diverse collection of 26 regression datasets from the UCI Machine Learning Repository [172], a data repository at the University of Porto [256], and a data repository at the University of Florida [273]. The datasets have sizes in the hundreds to tens of thousands and a number of features typically in the tens.

As described in Section 8.6, we split each dataset three ways into training, validation and testing (50/25/25%). We tuned the hyperparam-

ters 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.

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 8.6. 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 9.4.

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 10.1 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.

The mean out-of-sample R^2 across all datasets is shown in Figure 10.16 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 10.2, 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

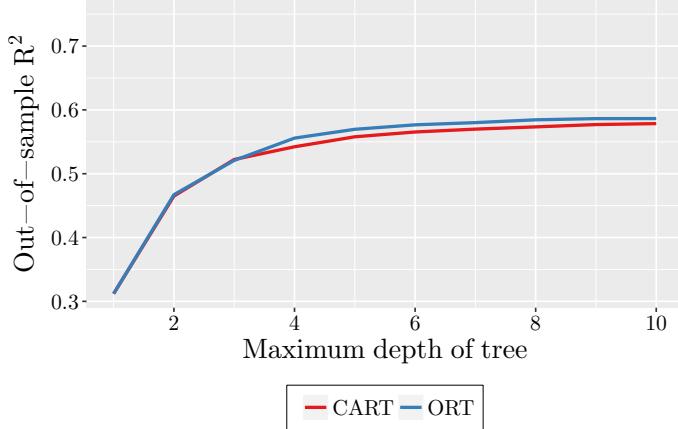
Table 10.1: Full results for CART and ORT at depth 10. The best method for each dataset is indicated in bold. Positive improvements are highlighted in blue, and negative in red.

Name	Dataset		Mean out-of-sample R^2		Mean improvement
	n	p	CART	ORT	
abalone	4176	7	0.460	0.457	-0.003 ± 0.013
airfoils-self-noise	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 ± 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 ± 0.084
computer-hardware	208	36	0.789	0.858	+0.069 ± 0.093
concrete-compressive	102	7	0.453	0.364	-0.088 ± 0.072
concrete-flow	102	7	0.208	0.251	+0.043 ± 0.028
concrete-slump	102	7	0.172	0.217	+0.045 ± 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 ± 0.003
friedman-artificial	40767	10	0.846	0.852	+0.006 ± 0.001
housing	505	13	0.705	0.750	+0.045 ± 0.058
hybrid-price	152	3	0.512	0.464	-0.048 ± 0.057
kin8nm	8191	8	0.442	0.498	+0.056 ± 0.014
lpga-2008	156	6	0.594	0.624	+0.030 ± 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 ± 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 ± 0.003
yacht-hydrodynamics	307	6	0.988	0.991	+0.003 ± 0.002

Table 10.2: 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 ± 0.006	0.7152
3	0.522	0.521	-0.001 ± 0.007	0.8298
4	0.542	0.556	+0.014 ± 0.008	0.0820
5	0.558	0.570	+0.012 ± 0.008	0.1507
6	0.565	0.577	+0.011 ± 0.008	0.1681
7	0.570	0.580	+0.010 ± 0.008	0.2034
8	0.573	0.584	+0.011 ± 0.008	0.1607
9	0.577	0.586	+0.009 ± 0.008	0.2330
10	0.579	0.586	+0.008 ± 0.008	0.3086

Figure 10.16: Mean out-of-sample R^2 for each method across all 26 datasets.



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.

All methods

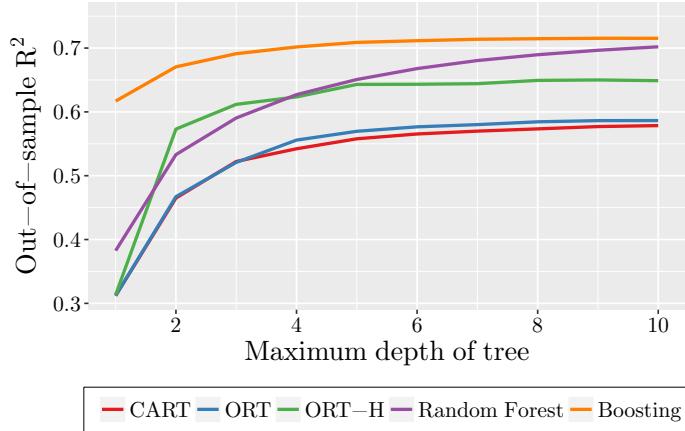
Next, we present a performance comparison of all methods considered. In addition to CART and ORT, we include results for ORT-H to investigate the impact of allowing hyperplane splits, as well as random forests and gradient boosted trees to compare our methods against those with state-of-the-art regression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

Figure 10.17 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 9.3 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 state-of-the-art performance, random forests and boosting. We can see that the strongest performer at all depths is boosting, improving

Figure 10.17: Mean out-of-sample R^2 for each method across all 26 datasets.



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 9.5. Table 10.3 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 10.4 shows the significance of the differences between each pair of methods, again following the procedure in Section 9.5. We can see that both boosting/random forests and ORT/CART do not have statistically significant differences in performance, 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 perfor-

Table 10.3: 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)
airfoils-self-noise	0.746 (4)	0.736 (5)	0.806 (3)	0.832 (2)	0.844 (1)
auto-mpg	0.816 (4)	0.814 (5)	0.863 (3)	0.910 (2)	0.941 (1)
automobile	0.771 (3)	0.771 (4)	0.766 (5)	0.847 (2)	0.849 (1)
cart-artificial	0.581 (4)	0.590 (3)	0.551 (5)	0.732 (2)	0.736 (1)
communities-and-crime	0.948 (1.5)	0.948 (1.5)	0.948 (4)	0.945 (5)	0.948 (3)
computer-hardware	0.409 (4)	0.507 (3)	0.390 (5)	0.702 (1)	0.679 (2)
concrete-compressive	0.789 (5)	0.858 (3)	0.797 (4)	0.929 (1)	0.929 (2)
concrete-flow	0.453 (4)	0.364 (5)	0.569 (3)	0.690 (2)	0.792 (1)
concrete-slump	0.208 (5)	0.251 (4)	0.380 (3)	0.437 (1)	0.396 (2)
cpu-act	0.172 (5)	0.217 (4)	0.570 (1)	0.359 (2)	0.272 (3)
cpu-small	0.970 (4)	0.966 (5)	0.975 (3)	0.981 (2)	0.985 (1)
elevators	0.959 (4)	0.959 (5)	0.967 (3)	0.974 (2)	0.979 (1)
friedman-artificial	0.686 (5)	0.689 (4)	0.874 (2)	0.815 (3)	0.877 (1)
housing	0.846 (5)	0.852 (4)	0.920 (2)	0.896 (3)	0.952 (1)
hybrid-price	0.705 (5)	0.750 (4)	0.753 (3)	0.852 (2)	0.867 (1)
kin8nm	0.512 (3)	0.464 (5)	0.509 (4)	0.615 (1)	0.577 (2)
lpga-2008	0.442 (5)	0.498 (4)	0.752 (2)	0.659 (3)	0.776 (1)
lpga-2009	0.594 (5)	0.624 (4)	0.647 (3)	0.770 (2)	0.775 (1)
parkinsons-motor	0.802 (4)	0.786 (5)	0.848 (3)	0.890 (1)	0.885 (2)
parkinsons-total	0.159 (4)	0.159 (5)	0.241 (3)	0.333 (2)	0.351 (1)
vote-for-clinton	0.144 (5)	0.163 (4)	0.283 (3)	0.334 (2)	0.361 (1)
wine-quality-red	0.315 (4)	0.286 (5)	0.354 (3)	0.408 (1)	0.404 (2)
wine-quality-white	0.286 (4)	0.263 (5)	0.299 (3)	0.429 (1)	0.394 (2)
yacht-hydrodynamics	0.279 (5)	0.282 (4)	0.300 (3)	0.442 (2)	0.497 (1)
Average rank	4.250	4.173	3.154	2.000	1.423

Table 10.4: 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. 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.

10.6 Concluding Remarks

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.

10.7 Notes and Sources

The material in this chapter is from Jack Dunn's PhD thesis [85].

Chapter 11

Optimal Regression Trees with Linear Predictions

It is a capital mistake to theorize before one has data.

– Arthur Conan Doyle, Author of Sherlock Holmes

Contents

- 11.1. ORT with Linear Predictions via MIO
- 11.2. ORT with Linear Predictions via Local Search
- 11.3. Experiments with Synthetic Datasets
- 11.4. Experiments with Real-World Datasets
- 11.5. Concluding Remarks
- 11.6. Notes and Sources

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 [221] 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 [71, 179, 143] 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. [160] 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.

11.1 ORT with Linear Predictions via MIO

In this section, we will modify the MIO formulation for Optimal Regression Trees in Section 10.2 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 (10.1). It is straightforward to extend this approach to use linear predictions instead. We will use the variables β_t and β_{0t} to track the prediction at each leaf, which takes the form

$$y(\mathbf{x}) = \beta_t^\top \mathbf{x} + \beta_{0t}$$

We then update the constraint (10.1) with the new prediction function:

$$f_i = \sum_{t \in \mathcal{T}_L} (\beta_t^\top \mathbf{x}_i + \beta_{0t}) z_{it}, \quad \forall i \in [n], \quad (11.1)$$

which linearizes in the same fashion to give

$$-M_f(1 - z_{ik}) \leq f_i - (\beta_t^\top \mathbf{x}_i + \beta_{0t}) \leq M_f(1 - z_{ik}), \quad \forall i \in [n], t \in \mathcal{T}_L, \quad (11.2)$$

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:

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

This implies that at optimality, f_i and $(\beta_t^\top \mathbf{x}_i + \beta_{0t})$ 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:

$$\frac{1}{L} \sum_{i=1}^n L_i + \alpha \cdot C + \lambda \sum_{t \in \mathcal{T}_L} \|\beta_t\|, \quad (11.3)$$

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 j th feature is used in the t th regression equation:

$$-M_r r_{jt} \leq \beta_{jt} \leq M_r r_{jt}, \quad \forall j \in [p], t \in \mathcal{T}_L \quad (11.4)$$

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:

$$\frac{1}{L} \sum_{i=1}^n L_i + \alpha \cdot C + \lambda \sum_{t \in \mathcal{T}_L} \sum_{j=1}^p r_{jt}, \quad (11.5)$$

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 10.2. For instance, the following is the ORT formulation (10.2) with linear predictions and using L_0 regularization:

$$\begin{aligned} \min \quad & \frac{1}{L} \sum_{i=1}^n L_i + \alpha \cdot C + \lambda \sum_{t \in \mathcal{T}_L} \sum_{j=1}^p r_{jt} \\ \text{s.t.} \quad & L_i \geq (f_i - y_i)^2, \quad \forall i \in [n], \end{aligned} \quad (11.6)$$

cite bertsimas and van
parrys here

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

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.

11.2 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 [47]. 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 [105], which solves the standard elastic net regression problem:

$$\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda P_\alpha(\boldsymbol{\beta}) \quad (11.7)$$

where

$$P_\alpha(\boldsymbol{\beta}) = (1 - \alpha) \frac{1}{2} \|\boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_1 \quad (11.8)$$

If we set $\alpha = 1$ to only use L_1 regularization (yielding the LASSO [251]), and denote the fitted values with $f_i = \beta_0 + \mathbf{x}_i^\top \boldsymbol{\beta}$, we can rewrite the problem in a simpler form as

$$\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{2n} \sum_{i=1}^n (y_i - f_i)^2 + \lambda \|\boldsymbol{\beta}\|_1 \quad (11.9)$$

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 (11.6) with L_1 regularization. Our overall objective is of the form

$$\min_{\hat{L}} \frac{1}{L} \sum_{i=1}^n (y_i - f_i)^2 + \lambda \sum_{t \in \mathcal{T}_L} \|\boldsymbol{\beta}_t\|_1, \quad (11.10)$$

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

$$\min_{t \in \mathcal{T}_L} \sum_{i \in \mathcal{I}_t} \left(\frac{1}{\hat{L}} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \lambda \|\beta_t\|_1 \right). \quad (11.11)$$

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

$$\min_{i \in \mathcal{I}_t} \frac{1}{\hat{L}} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \lambda \|\beta_t\|_1, \quad (11.12)$$

which we seek to solve using the GLMNET algorithm. We can do so by rewriting this problem into the same form as (11.9):

$$\min_{i \in \mathcal{I}_t} \frac{1}{2|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} (y_i - f_i)^2 + \frac{\lambda \hat{L}}{2|\mathcal{I}_t|} \|\beta_t\|_1, \quad (11.13)$$

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 (11.10).

The local search therefore proceeds as for constant-prediction regression trees in Section 10.3, except that each time we are required to evaluate the loss of the tree, we update the current regression predictions in each leaf by performing coordinate-descent cycles for the problems (11.13) 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.

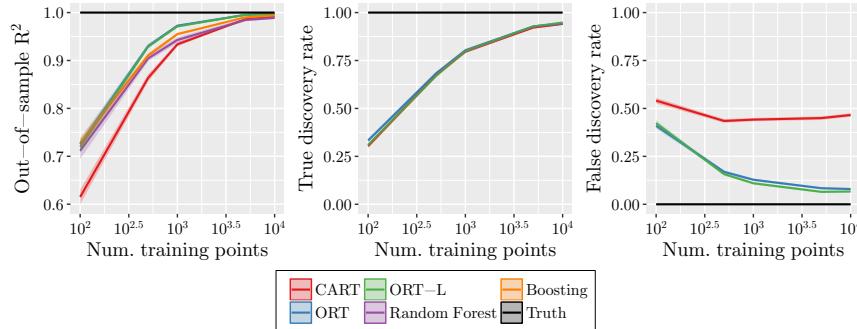
talk about complexity?

Hyperparameter tuning

Using linear regression predictions in each leaf introduces the regularization parameter λ as a new tuning parameter that needs to be specified. It would be possible to use the same procedure as in Algorithm 8.7 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 8.7. We then fix this depth, and tune λ using Algorithm 8.7 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.

Figure 11.1: Synthetic experiments showing the effect of training set size.



11.3 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 10.4, with the addition of the Optimal Regression Trees with linear predictions, both with (ORT-HL) and without (ORT-L) hyperplane splits. We used Algorithm 8.7 to tune ORT-L and ORT-HL, 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.

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 11.1. We can see that in all three metrics, ORT and ORT-L

Figure 11.2: Synthetic experiments showing the effect of number of features.

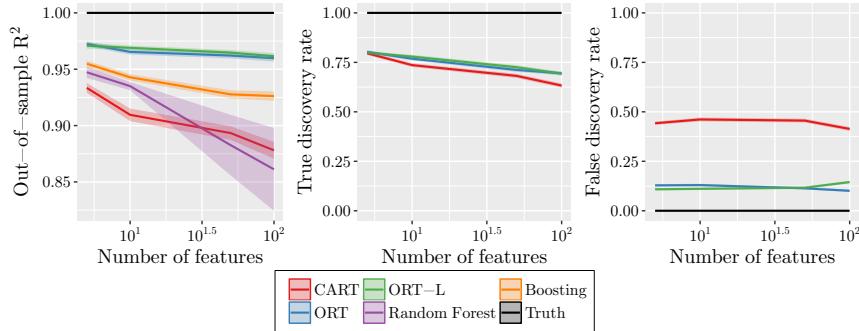
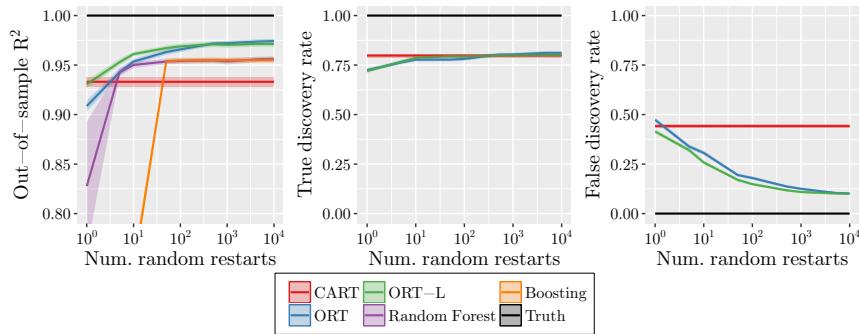


Figure 11.3: Synthetic experiments showing the effect of number of random restarts.



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 11.2 shows the results. Again we see that the performance of ORT and ORT-L is similar, with ORT-L actually performing slightly 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 11.3. ORT and ORT-L again perform similarly, although ORT-L has a sizable advantage

Figure 11.4: Synthetic experiments showing the effect of feature noise.

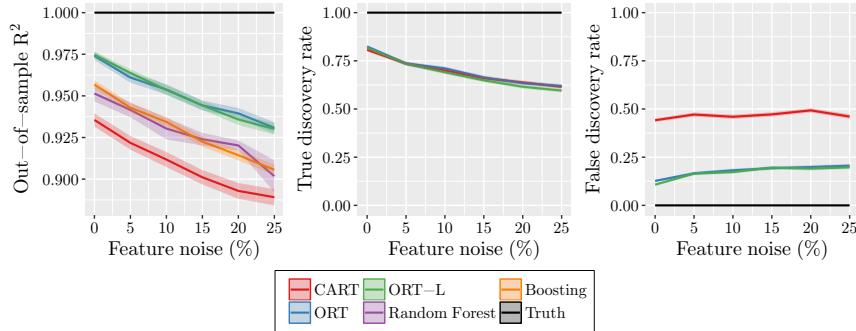
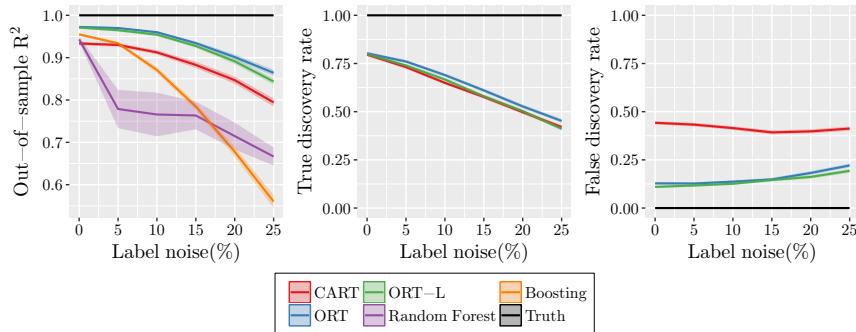


Figure 11.5: Synthetic experiments showing the effect of label noise.



in R^2 for very small numbers of random restarts, as well as a slightly lower FDR over all values.

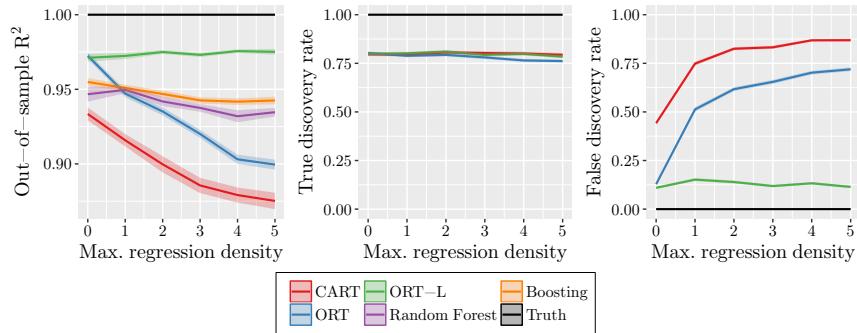
The effect of noise in the training data

The final two experiments show the effect of noise in the training set on the performance of the methods. Figures 11.4 and 11.5 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

Figure 11.6: Synthetic experiments showing the effect of regression density.



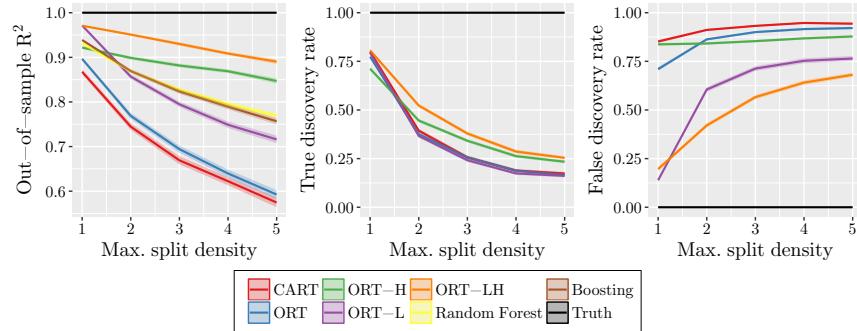
power of ORT-L, it is not at risk of overfitting data that are described by simpler models. In particular, our validation procedure is able to correctly identify both the correctly-sized tree but also the correct degree of regularization in the linear predictions for a given dataset.

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 11.6 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

Figure 11.7: Synthetic experiments showing the effect of split density.



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.

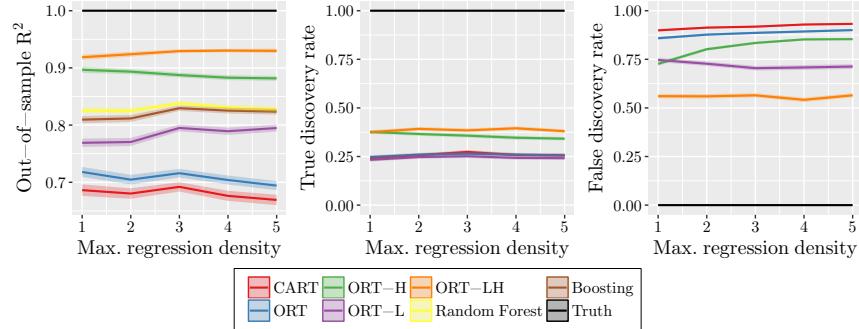
Ground truth trees with linear predictions and hyperplane splits

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 11.7. 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 11.6. We also see that ORT-LH performs 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 than 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 11.8 shows the results. Again we see that ORT-LH is the strongest performing method, followed by ORT-H. The results are largely the same across all regression

Figure 11.8: Synthetic experiments showing the effect of regression density.



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.

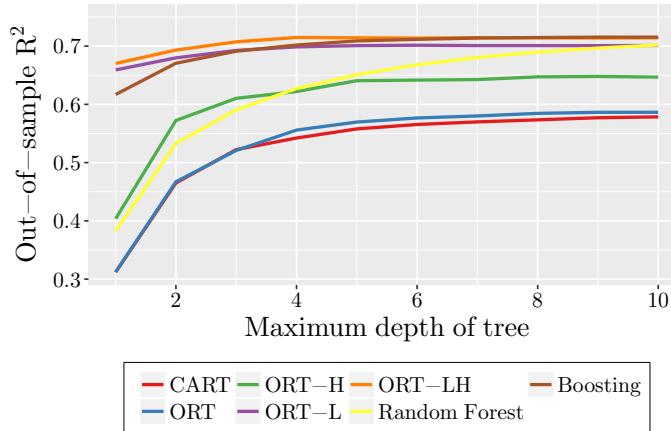
Secondly, we found that the increased flexibility of trees with linear predictions and/or hyperplane splits did not lead them to overfit in scenarios where the true model was of a simpler class. For instance, ORT-L did not overfit when the true predictions were constant, and ORT-LH did not overfit when 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.

11.4 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

Figure 11.9: Mean out-of-sample R^2 for each method across all 26 datasets.



same set of real-world datasets as in Section 10.5. 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 8.6. 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 11.3.

Results

In Figure 11.9 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 10.17 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 11.1 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

Table 11.1: 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)
airlines	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

as the average rank of each method. We can see that the results for CART, ORT and ORT-H mirror those in Section 10.5—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 11.2 shows the p-values for pairwise comparisons between each pair of methods, appropriately adjusted to account for multiple comparisons as described in Section 9.5. 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 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.

Table 11.2: 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

11.5 Concluding Remarks

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 8–11 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.

11.6 Notes and Sources

The material in this chapter is from Jack Dunn’s PhD thesis [85].

Chapter 12

Optimal Trees and Neural Networks

For artificial intelligence to thrive, it must explain itself. If it cannot, who will trust it?

– The Economist, February 15, 2018

Contents

- 12.1. Mathematical Formulation of Neural Networks
- 12.2. FNNs and Optimal Trees
- 12.3. CNNs and Optimal Trees
- 12.4. RNNs and Optimal Trees
- 12.5. Transforming OCT-Hs into Classification NNs
- 12.6. Computational Results
- 12.7. Concluding Remarks
- 12.8. Notes and Sources

Neural networks, a supervised learning technique, have become one of the most widely used machine learning techniques today. Generally, a neural network's architecture is defined by

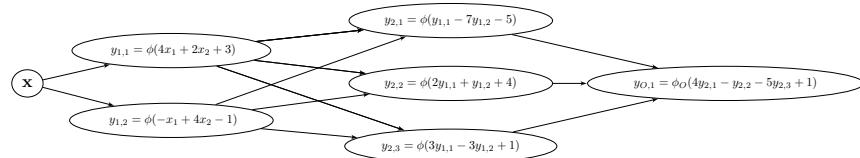
- L hidden layers, indexed $\ell = 1, \dots, L$, and one output layer.
- Hidden layer ℓ consisting of N_ℓ nodes, indexed $i = 1, \dots, N_\ell$.
- Some non-linear function $\phi(x)$ associated with the hidden layers.
- Some function $\phi_O(x)$ associated with the output layer.

An example of a feedforward neural network can be seen in Figure 12.1. It has 2 hidden layers, with $N_1 = 2$ nodes in the first hidden layer and $N_2 = 3$ nodes in the second. One of the major innovations that took place in neural networks was an increase in variety in the $\phi(\cdot)$ functions used. While earlier versions such as the Perceptron used an indicator function $\phi(x) = 1\{x \geq 0\}$, to try to capture the binary way neurons either fired or did not, some more recent choices such as $\phi(x) = \max(x, 0)$ have also been used to create effective neural networks with high predictive accuracy.

Learning using very large neural networks, called deep learning, has had some great successes across a variety of applications. For example, it revolutionized image recognition [243, 120, 131], with competitors in the ImageNet Large Scale Visual Recognition Challenge almost exclusively using deep learning in their programs [235, 163]. It has also been very successful in machine translation, and other forms of natural language processing — Google Translate improved dramatically when it began to use deep learning in its translation algorithms [192, 73, 8, 248]. Additionally, speech recognition technology has markedly improved with the inclusion of deep learning [115, 121].

These examples, however, represent only a fraction of the applications neural networks are used in. With high profile successes, over 2 million results in a search on Google scholar on the topic “neural network articles” [118], and influential articles in the field garnering thousands of citations, it is clear that neural networks are an integral part of the field of artificial intelligence nowadays. However, neural networks face some challenges alongside their strengths and successes. They rely on heuristics in their training process, like dropout [245, 277, 135] and early

Figure 12.1: An example of a feedforward neural network.



stopping [227, 281]. While neural networks often work well, it is unclear when they work well, why they work well, and if they do not work well how to improve them. Importantly, given that they have thousands to tens of thousands of parameters, they are not interpretable by humans.

In this chapter, we investigate the modeling power of neural networks in comparison with OCT-Hs and ORT-Hs. We prove that a variety of neural networks (feedforward, convolutional and recurrent) can be transformed to classification and regression trees with hyperplanes with the same accuracy in the training set, showing that OCT-Hs and ORT-Hs are at least as powerful as neural networks. Conversely, a given classification tree with hyperplanes can be transformed to a classification neural network with the same accuracy in the training set, showing that these neural networks are at least as powerful as OCT-Hs. Consequently, we show that OCT-Hs and neural networks are equivalent in terms of modeling power. Given that our constructions necessitate the construction of decision trees of significant depth, we explored the practical implication of these findings. We report a comparison of OCT-Hs and neural networks on seven well known data sets and show that the two methods exhibit remarkably close accuracy.

Tables 12.1 and 12.2 summarize the key results of this chapter for classification and regression problems, respectively, showing the parameters of the construction and the section containing the construction.

12.1 Mathematical Formulation of Neural Networks

In this section, we describe the mathematical structure of the different types of neural networks we consider in this chapter.

Feedforward Neural Networks

A classification feedforward neural network (FNN) is trained on input and output pairs (\mathbf{x}_j, o_j) , $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $o_j \in \{1, \dots, q\}$, $j \in [N]$. The output o_j represents the class to which the point \mathbf{x}_j belongs. The characteristics of FNNs are:

- There are L hidden layers. Hidden layer $\ell \in [L]$ consists of N_ℓ nodes. Node $n_{\ell,i}$, $i \in [N_\ell]$, in hidden layer ℓ is characterized by a vector $\mathbf{W}_{\ell,i} \in \mathbb{R}^{N_{\ell-1}}$ and scalar $b_{\ell,i} \in \mathbb{R}$. The node computes

$$y_{\ell,i} = \phi(\mathbf{W}_{\ell,i}^T \mathbf{y}_{\ell-1} + b_{\ell,i}), \quad (12.1)$$

where $\phi(x)$ is a nonlinear function, and $\mathbf{y}_{\ell-1}$ is the vector of outputs of the hidden layer $\ell - 1$. We define $\mathbf{y}_0 \triangleq \mathbf{x}$, the input of the FNN.

- There is one output layer that consists of q nodes. Node i , $i \in [q]$ in the output layer is characterized by $\mathbf{W}_{O,i} \in \mathbb{R}^{N_L}$ and scalar $b_{O,i} \in \mathbb{R}$.

Table 12.1: Summary of the relationship between classification NNs and OCT-Hs.

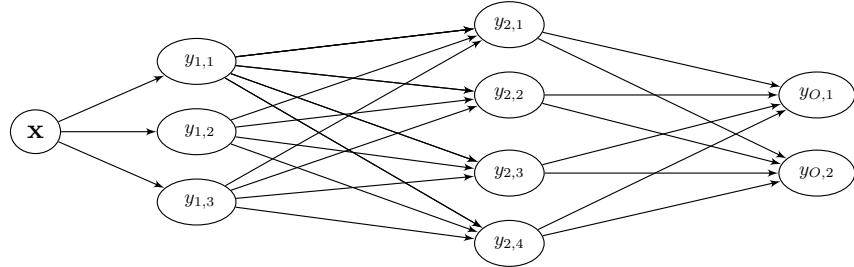
Model	Parameters	New Model	Parameters	Section
Classification Feedforward Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	OCT-H	<ul style="list-style-type: none"> • Depth N_1 	12.2
Classification Feedforward Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \max(x, 0)$ 	OCT-H	<ul style="list-style-type: none"> • Depth $\sum_{\ell=1}^L N_\ell$ 	12.2
Classification Convolutional Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	OCT-H	<ul style="list-style-type: none"> • Depth N_1 	12.3
Classification Convolutional Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \max(x, 0)$ 	OCT-H	<ul style="list-style-type: none"> • Depth $\sum_{\ell=1}^L N_\ell$ 	12.3
Classification Recurrent Neural Network	<ul style="list-style-type: none"> • Input of length T^* • 1 layer • N_1 nodes in layer • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	OCT-H	<ul style="list-style-type: none"> • Depth $T^* \times N_1$ 	12.4
Classification Recurrent Neural Network	<ul style="list-style-type: none"> • Input of length T^* • 1 layer • N_1 nodes in 1st layer • $\phi(x) = \max(x, 0)$ 	OCT-H	<ul style="list-style-type: none"> • Depth $q - 1 + T^* \times N_1$ 	12.4
OCT-H	<ul style="list-style-type: none"> • N_1 splits • N_2 leaf nodes • q classification values 	Classification Feedforward Neural Network	<ul style="list-style-type: none"> • 2 layers • N_1 nodes in 1st layer • N_2 nodes in 2nd • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	12.5

Table 12.2: Summary of the relationship of regression NNs and ORT-Hs.

Model	Parameters	New Model	Parameters	Section
Regression Feedforward Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	ORT-H	<ul style="list-style-type: none"> • Depth N_1 	12.2
Regression Feedforward Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \max(x, 0)$ 	ORT-H	<ul style="list-style-type: none"> • Depth $q - 1 + \sum_{\ell=1}^L N_\ell$ 	12.2
Regression Convolutional Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	ORT-H	<ul style="list-style-type: none"> • Depth N_1 	12.3
Regression Convolutional Neural Network	<ul style="list-style-type: none"> • L layers • N_ℓ nodes in layer ℓ • $\phi(x) = \max(x, 0)$ 	ORT-H	<ul style="list-style-type: none"> • Depth $q - 1 + \sum_{\ell=1}^L N_\ell$ 	12.3
Regression Recurrent Neural Network	<ul style="list-style-type: none"> • Input of length T^* • 1 layer • N_1 nodes in 1st layer • $\phi(x) = \mathbb{1}\{x \geq 0\}$ 	ORT-H	<ul style="list-style-type: none"> • Depth $T^* \times N_1$ 	12.4
Regression Recurrent Neural Network	<ul style="list-style-type: none"> • Input of length T^* • 1 layer • N_1 nodes in 1st layer • $\phi(x) = \max(x, 0)$ 	ORT-H	<ul style="list-style-type: none"> • Depth $T^* \times N_1$ 	12.4

Figure 12.2: An example of a classification feedforward neural network with:

- $y_{1,j} = \phi(\mathbf{W}'_{1,j}\mathbf{x} + b_{1,j})$, $j = 1, 2, 3$,
- $y_{2,j} = \phi(\mathbf{W}'_{2,j}\mathbf{y}_1 + b_{2,j})$, $j = 1, \dots, 4$,
- $y_{O,j} = \phi_O(\mathbf{W}'_{O,j}\mathbf{y}_2 + b_{O,j})$, $j = 1, 2$.



Unlike in the hidden layers, the output layer's activation function $\phi_O(x)$ is a vector valued function, where

$$y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + \mathbf{b}_O)_i, \quad i \in [q]. \quad (12.2)$$

For ease of notation in the diagrams, we will define

$$\phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + \mathbf{b}_O)_i = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + b_O)_i, \quad i \in [q]. \quad (12.3)$$

Note that $\phi_O(x)$ need not be the same as $\phi(x)$ applied element-wise to the vector \mathbf{y}_L .

- The final prediction of the network is found by calculating $k = \text{arg-lex-max}_{i \in [q]}(y_{O,i})$. The lexicographic maximum means that in case of a tie, the smallest index k is our choice. We then use k as our predicted class value.

An example of a classification feedforward neural network is depicted in Figure 12.2. Here, we have 2 hidden layers, with 3 nodes in the first hidden layer, 4 nodes in the second, and 2 nodes in the output layer. Also note that each node in this network has its own unique weights $\mathbf{W}_{\ell,i}$ and $b_{\ell,i}$ (which we have to solve for in the training process), and that the nodes in one layer have directed edges leading to all the nodes in the next layer (a trait known as being “fully connected”).

Two potential choices for the activation function $\phi(\cdot)$ are the perceptron activation function, where

$$\phi(x) = \mathbb{1}\{x \geq 0\}, \quad (12.4)$$

and the rectified linear unit or ReLU, where

$$\phi(x) = \max(x, 0). \quad (12.5)$$

If (12.4) is chosen as the activation function, then $(\phi_O(\mathbf{x}))_i = \mathbb{1}\{x_i \geq 0\}$. If (12.5) is chosen as the activation function, then $\phi_O(\mathbf{x})$ is defined as

$$(\phi_O(\mathbf{x}))_i = \begin{cases} 1, & \text{where } i = \arg \max_{i \in [N_0]} x_i, \\ 0, & \text{otherwise.} \end{cases} \quad (12.6)$$

A regression FNN is defined in much the same way as the above, with a two minor differences. First, the training data pairs are $(\mathbf{x}_j, \mathbf{o}_j)$, $j \in [N]$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$.

When using neural networks, the standard goal is to solve for weight matrices and bias vectors \mathbf{W}_ℓ and \mathbf{b}_ℓ , $\ell \in [L] \cup \{O\}$. This is usually done by minimizing some loss function using stochastic gradient descent [49, 168], and is typically augmented with techniques like early stopping [227, 281] and dropout [245, 277, 135]. Due to the heuristic nature of these methods, we are not guaranteed an optimal solution for \mathbf{W}_ℓ and \mathbf{b}_ℓ , $\ell \in [L] \cup \{O\}$.

Convolutional Neural Networks

A convolutional neural network (CNN) [239, 115] is designed to be especially effective at classifying images, so their architecture is meant to exploit the traits of image data. CNNs use three types of hidden layers: convolutional, pooling and regular layers. They are trained on input and output pairs (\mathbf{x}_j, o_j) , $j \in [N]$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $o_j \in \{1, \dots, q\}$, $j \in [N]$. The output o_j represents the class to which the point \mathbf{x}_j belongs. Their characteristics are as follows:

- There are L hidden layers in the network. The first L_c are pairs of convolutional layers and pooling layers, while the remaining $L - L_c$ are regular hidden layers.
- The convolutional and the regular hidden layers each have N_ℓ nodes, $\ell = 1, \dots, L_c, L_c + 1, \dots, L$. Node $n_{\ell,i}$, $i \in [N_\ell]$, $\ell \in [L]$ is characterized by a vector $\mathbf{W}_{\ell,i} \in \mathbb{R}^{N_{\ell-1}}$ and scalar $b_{\ell,i} \in \mathbb{R}$. The node computes

$$y_{\ell,i} = \begin{cases} \phi(\mathbf{W}_{\ell,i}^T \mathbf{P}_{\ell-1} + b_{\ell,i}), & \text{if } \ell = 1, \dots, L_c, \\ \phi(\mathbf{W}_{\ell,i}^T \mathbf{y}_{\ell-1} + b_{\ell,i}), & \text{if } \ell = L_c + 1, \dots, L, \end{cases} \quad (12.7)$$

where $\phi(x)$ is a nonlinear function, $\mathbf{P}_{\ell-1}$ is the vector of outputs of the previous pooling layer, $\ell = 1, \dots, L_c$ and $\mathbf{y}_{\ell-1}$ is the vector of outputs of the previous regular hidden layer $\ell = L_c + 1, \dots, L$.

- We define $\mathbf{P}_0 \triangleq \mathbf{x}$
- The pooling layers all have R_ℓ nodes, $\ell = 1, \dots, L_c$, with the property that $k_\ell = \frac{N_\ell}{R_\ell}$ is an integer. Node $p_{\ell,i}$, $i \in [R_\ell]$ in a pooling layer computes

$$P_{\ell,i} = \psi(y_{\ell,m}, m \in S_{\ell,i}),$$

where $\psi(\cdot)$ is a nonlinear function, \mathbf{y}_ℓ is the vector of outputs of the previous convolutional layer, and $S_{\ell,i}$ is the collection of indices that affect the computation of $P_{\ell,i}$. The sets $S_{\ell,i}$ have the following properties:

1. $\bigcup_{i=1}^{R_\ell} S_{\ell,i} = \{1, \dots, N_\ell\}$
2. $S_{\ell,i} \cap S_{\ell,j} = \emptyset$
3. $|S_{\ell,i}| = k_\ell, i \in [R_\ell]$

The most commonly used pooling layer type is the max pooling layer that calculates:

$$P_{\ell,i} = \max_{m \in S_{\ell,i}} (y_{\ell,m}), \quad i \in [R_\ell]. \quad (12.8)$$

- The output layer consists of q nodes. A node in the output layer is characterized by $\mathbf{W}_{O,i} \in \mathbb{R}^{N_L}$ and scalar $b_{O,i} \in \mathbb{R}$. The node computes $y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_L + b_{O,i})$ as defined in Eq. (12.2), where $\phi_O(x)$ is some function, which need not be $\phi(x)$ applied element-wise to an input vector.
- The final prediction of the network is found by calculating $k = \arg\text{-lex}\text{-max}_{i \in [q]} (y_{O,i})$. The lexicographic maximum means that in case of a tie, the smallest index k is our choice. We then use k as our predicted class value.

A regression CNN is defined in much the same way as the above, with a two minor differences. First, the training data pairs are $(\mathbf{x}_j, \mathbf{o}_j)$, $j \in [N]$, where $\mathbf{x}_j \in \mathbb{R}^{N_0}$ and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are specialized to be able to handle sequential input data [176]. In general, it is expected that the data have the same dimensions at each step in the sequence, but that the sequence can be of arbitrary length. For exposition reasons we assume both that the data have the same dimensions at each step in the sequence and that every potential input sequence has length T^* . The way that a simple RNN works is that the network has a single hidden layer that processes the data in a sequential manner over a series of time steps $t \in [T^*]$. It “remembers” past inputs by using the hidden layer output at time step t as an input into the hidden layer at time step $t+1$. As a result of this, the network structure of RNNs contains self loops, unlike FNNs and CNNs.

At the final time step T^* , and given output function $\phi_O(\cdot)$ and output weights $\mathbf{W}_O \in \mathbb{R}^{n_1 \times q}$ and $\mathbf{b}_O \in \mathbb{R}^q$, we then calculate \mathbf{y}_O , where $y_{O,i} = \phi_O((\mathbf{W}_O \mathbf{y}_{T^*,1} + \mathbf{b}_O)_i)$ for $i \in [q]$. This classification prediction can be

viewed in some cases as the network predicting the next value in the given sequence.

A recurrent neural network is trained on the input sequence and output pair $(\mathbf{x}_{t,j}, o_j)$, $j \in [N]$, where $\mathbf{x}_t \in \mathbb{R}^{N_0}$, $t \in [T^*]$, and $o_j \in \{1, \dots, q\}$ is the class the sequence $(\mathbf{x}_{t,j})_{t \in [T^*]}$ is in. Their characteristics are as follows:

- There is one hidden layer and one output layer. The hidden layer contains N_1 nodes, $i \in [N_1]$. Node $n_{1,i}$ is characterized by vectors $\mathbf{W}_{g,i} \in \mathbb{R}^{N_0}$ and $\mathbf{W}_{h,i} \in \mathbb{R}^{N_1}$, and scalar $b_{1,i}$. It computes

$$y_{t,1,i} = \phi(\mathbf{W}_{g,i}^T \mathbf{x}_t + \mathbf{W}_{h,i} \mathbf{y}_{t-1,1} + b_{1,i}), \quad (12.9)$$

where $\phi(x)$ is some non-linear function, and $\mathbf{y}_{t-1,1}$ is the vector of outputs of the hidden layer in the previous time step.

- We define $\mathbf{y}_{0,1}$ as the zero vector $\mathbf{0} \in \mathbb{R}^{N_1}$.
- The output layer consists of q nodes, $i \in [q]$. A node in the output layer is characterized by $\mathbf{W}_{O,i} \in \mathbb{R}^{N_1}$ and scalar $b_{O,i} \in \mathbb{R}$. It computes $y_{O,i} = \phi_O(\mathbf{W}_{O,i}^T \mathbf{y}_{T^*,1} + b_{O,i})$ as defined in Eq. (12.3), where $\phi_O(x)$ is some function that need not be the same as $\phi(x)$ applied element-wise to the vector \mathbf{y}_L .
- The final prediction of the network is found by calculating $k = \arg\text{-lex}\text{-max}_{i \in [q]}(y_{O,i})$. The lexicographic maximum means that in case of a tie, the smallest index k is our choice. We then use k as our predicted class value.

A regression RNN is defined in much the same way as the above, with two minor differences. First, the training data inputs are $(\mathbf{x}_{t,j}, \mathbf{o}_j)$, $j \in [N]$, where $\mathbf{x}_t \in \mathbb{R}^{N_0}$, $t \in [T^*]$, and $\mathbf{o}_j \in \mathbb{R}^q$. Second, $\phi_O(\mathbf{x})$ is the identity function, so the network simply outputs $\mathbf{W}_{O,i}^T \mathbf{y}_{T^*,1} + \mathbf{b}_O$.

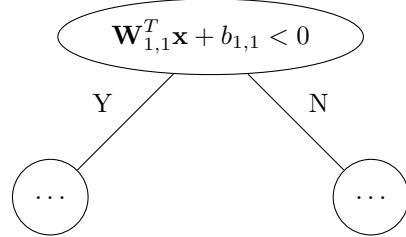
12.2 FNNs and Optimal Trees

In this section, we construct an OCT-H (ORT-H) that can classify training data at least as well as a classification (regression) feedforward neural network with perceptron or rectified linear unit activation functions.

Perceptron Classification FNNs and OCT-Hs

The key result of this section is as follows:

Theorem 12.1. *An OCT-H with maximum depth N_1 can classify the data in a training set at least as well as a given classification FNN with the perceptron activation function (12.4) and N_1 nodes in the first hidden layer.*

Figure 12.3: The first split of decision tree \mathcal{T}_1 .

Proof. Our proof is constructive. We are given a FNN \mathcal{N}_1 with the following characteristics:

- The perceptron activation function as defined in (12.4).
- Output function $\phi_O(\mathbf{x}) : [0, 1]^q \rightarrow [0, 1]^q$.
- L hidden layers and one output layer, indexed $\ell \in [L] \cup \{O\}$.
- N_ℓ nodes in each layer, indexed $i \in [N_\ell]$.
- q nodes in the output layer, indexed $i \in [q]$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.

We construct a classification tree \mathcal{T}_1 with hyperplane splits of maximum depth N_1 that makes the same predictions as \mathcal{N}_1 . This construction relies on the fact that a FNN with the perceptron activation function and N_1 nodes in the first hidden layer has at most 2^{N_1} distinct output values, which we can assign to the 2^{N_1} leaf nodes of \mathcal{T}_1 . It follows that an OCT-H of maximum depth N_1 has at least the same classification accuracy as \mathcal{N}_1 .

Given the inequality from the first node in the first hidden layer of \mathcal{N}_1 defined by the weight vector $\mathbf{W}_{1,1}$ and bias scalar $b_{1,1}$, we define the first split of \mathcal{T}_1 as

$$\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1} < 0, \quad (12.10)$$

which results in the simple split seen in Figure 12.3.

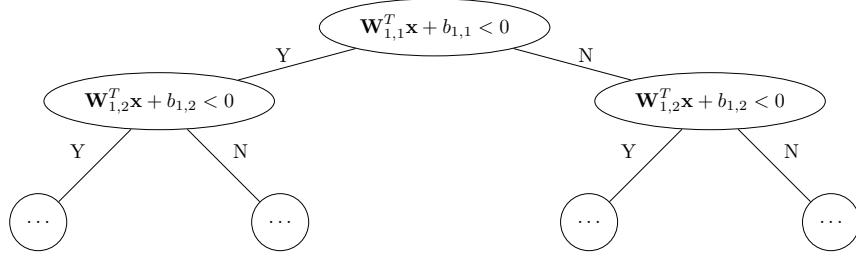
Independent of whether inequality (12.10) is satisfied or not, the second split is given by

$$\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2} < 0. \quad (12.11)$$

Figure 12.4 provides a visualization of the new branches we added to the tree in Figure 12.3.

We continue this process for all N_1 nodes in the first hidden layer, building a decision tree of depth N_1 , with every split at depth N_1 being given by

$$\mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1} < 0. \quad (12.12)$$

Figure 12.4: The first two depths of tree \mathcal{T}_1 .

Having defined the splits of the tree \mathcal{T}_1 , we now outline how to find the class value associated with each of the tree's leaves. Suppose that input \mathbf{x} is assigned to the r th leaf node of tree \mathcal{T}_1 , $r \in [2^{N_1}]$. Define $\mathcal{I}_{1,r}$ as

$$\mathcal{I}_{1,r} = \{i \mid \mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} \geq 0\}, \quad (12.13)$$

which is the set of all depths i of tree \mathcal{T}_1 where the inequality $\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} \geq 0$ is satisfied by an input \mathbf{x} sorted to the r th leaf node. Likewise define $\mathcal{I}_{2,r}$ as

$$\mathcal{I}_{2,r} = \{i \mid \mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i} < 0\}. \quad (12.14)$$

We know that in the neural network the first hidden layer outputs are

$$\begin{aligned} 1\{\mathbf{W}'_{1,i}^T \mathbf{x} + b_{1,i} \geq 0\} &= 1 \text{ for } i \in \mathcal{I}_{1,r} \text{ and} \\ 1\{\mathbf{W}_{2,i}^T \mathbf{x} + b_{2,i} \geq 0\} &= 0 \text{ for } i \in \mathcal{I}_{2,r}. \end{aligned}$$

To complete the construction of \mathcal{T}_1 , we need to assign a classification value for every leaf of \mathcal{T}_1 . Given an input \mathbf{x} of \mathcal{N}_1 , there are 2^{N_1} possible binary vectors that the first hidden layer of \mathcal{N}_1 could output. These 2^{N_1} vectors, by our construction of \mathcal{T}_1 , exactly correspond to the 2^{N_1} leaves of \mathcal{T}_1 . Given the output $\mathbf{y}_1(\mathbf{x})$ of the first hidden layer, the final prediction of \mathcal{N}_1 will be $k(\mathbf{y}_1)$, which is calculated deterministically given the \mathbf{y}_1 vector and the $\mathbf{W}_{\ell,i}$, $b_{\ell,i}$ values by using the process outlined in Section 12.1. In every node r of the tree we assign the classification value $k(\mathbf{y}_1)$.

We next show that the output of \mathcal{T}_1 is the same as the output of \mathcal{N}_1 for input data point \mathbf{x} . To see this, if \mathbf{x} is input into \mathcal{N}_1 , the first hidden layer outputs \mathbf{y}_1 , resulting in the final network output $k(\mathbf{y}_1)$. However, in the decision tree, \mathbf{x} is assigned to the leaf node corresponding to \mathbf{y}_1 , and is once again assigned output value $k(\mathbf{y}_1)$ by construction. Thus, for a given data point \mathbf{x} , the network and the tree predict the same classification value.

Since an OCT-H does at least as well as \mathcal{T}_1 in classifying the training data, it must do at least as well as \mathcal{N}_1 too. Thus, by construction, we have that an optimal decision tree with maximum depth N_1 can classify data in a training set at least as well as the given FNN with perceptron

activation function, $L \geq 1$ hidden layers, and N_1 nodes in the first hidden layer, completing the proof of the theorem. \square

We remark that

1. The construction of tree \mathcal{T}_1 is independent of L , the number of hidden layers.
2. While the output function $\phi_O(\mathbf{x})$ affects the values for classification, the construction of \mathcal{T}_1 is not affected by $\phi_O(\mathbf{x})$; only the output values of the leaves of \mathcal{T}_1 are affected by $\phi_O(\mathbf{x})$.

Perceptron Regression FNNs and ORT-Hs

The only difference between a regression FNN and a classification FNN in the construction of the network is that $\phi_O(\mathbf{y}_L) \in \mathbb{R}^q$. Because the first hidden layer of a regression FNN with the perceptron activation function can output at most 2^{N_1} unique vectors, there are only 2^{N_1} possible output values of the network. In this case, one can modify the proof in Section 12.2 by assigning these 2^{N_1} unique values to the leaf nodes of the decision tree in the place of classification values. With this adjustment, extending Theorem 12.1 to regression FNNs with perceptron activation functions is straightforward.

ReLU Classification FNNs and OCT-Hs

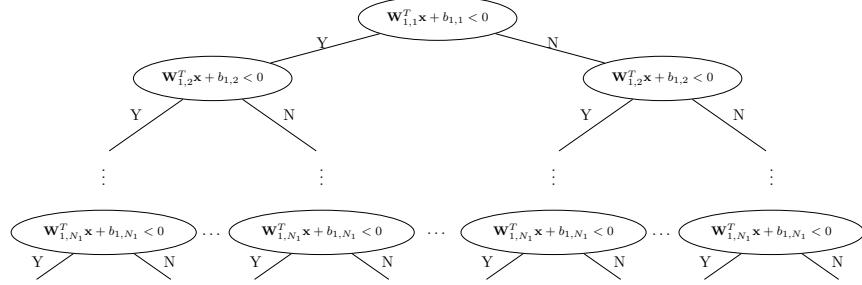
In this section, we show that given a classification FNN with rectified linear unit activation functions, we can construct a classification tree with hyperplane splits that can classify the given training data at least as well as the given FNN. The theorem is as follows.

Theorem 12.2. *An OCT-H with maximum depth $q - 1 + \sum_{\ell=1}^L N_\ell$ can classify data in a training set at least as well as a given classification FNN with the rectified linear unit activation function (12.5), L hidden layers, N_ℓ nodes in layer $\ell \in [L]$, and q nodes in the output layer.*

Proof. We are given a FNN \mathcal{N}_2 with the following characteristics:

- The rectified linear unit activation function, as defined in (12.5).
- L hidden layers and one output layer, indexed $\ell \in [L] \cup \{O\}$.
- N_ℓ nodes in each layer, indexed $i \in [N_\ell]$.
- q nodes in the output layer, indexed $i \in [q]$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.

Figure 12.5: The decision tree \mathcal{T}_2 we are building up to depth N_1 .



We construct a classification tree \mathcal{T}_2 with hyperplane splits of maximum depth $q - 1 + \sum_{\ell=1}^L N_\ell$ that makes the same predictions as \mathcal{N}_2 . It follows that an OCT-H of maximum depth $q - 1 + \sum_{\ell=1}^L N_\ell$ has at least the same classification accuracy as \mathcal{N}_2 .

We build the tree up to depth N_1 exactly the same way as in the proof in Section 12.2. This part of the tree is shown in Figure 12.5. This is the subtree that simulates the output of \mathcal{N}_2 after the first hidden layer.

After depth N_1 , there are 2^{N_1} branches, as shown in Figure 12.5. Note that there are 2^{N_1} possible output vectors \mathbf{y}_1 of the first layer of \mathcal{N}_2 ,

$$(0, \dots, 0)^T, (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}, \dots, 0)^T, \dots, (\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}, \dots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T,$$

as each node of the first layer \mathcal{N}_2 computes

$$\mathbf{y}_{1,i} = \max\{\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i}, 0\}, \quad i \in [N_1]. \quad (12.15)$$

These 2^{N_1} possible values of \mathbf{y}_1 correspond to the 2^{N_1} branches in the tree \mathcal{T}_2 shown in Figure 12.5. In each of these branches we have implicitly calculated the corresponding \mathbf{y}_1 values. For example, the first branch corresponds to $(0, \dots, 0)^T$, the second corresponds to $(0, \dots, 0, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T$, etc.

We then model the second layer of \mathcal{N}_2 by constructing after each branch a new subtree of depth N_2 as in Figure 12.5, but with the corresponding value of \mathbf{y}_1 playing the role of \mathbf{x} .

In the subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ in Figure 12.6 we substitute in the corresponding value of \mathbf{y}_1 as a linear function of \mathbf{x} . For example, if $\mathbf{y}_1 = (0, \dots, 0, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})$, then subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ is depicted in Figure 12.7.

Given that at each branch we know exactly \mathbf{y}_1 , as an explicit function of \mathbf{x} , $\mathcal{T}_{2,2}(\mathbf{y}_1)$ is a decision tree where all inequalities are explicitly written as linear functions of \mathbf{x} .

Continuing in this way we model the output vector \mathbf{y}_ℓ of the ℓ th hidden layer of \mathcal{N}_2 as a classification tree by propagating the values of $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_L$ as explicit linear functions of \mathbf{x} .

Figure 12.6: Subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ of depth N_2 is concatenated to the corresponding branch of the subtree depicted in Figure 12.5, resulting in a subtree of depth $N_1 + N_2$.

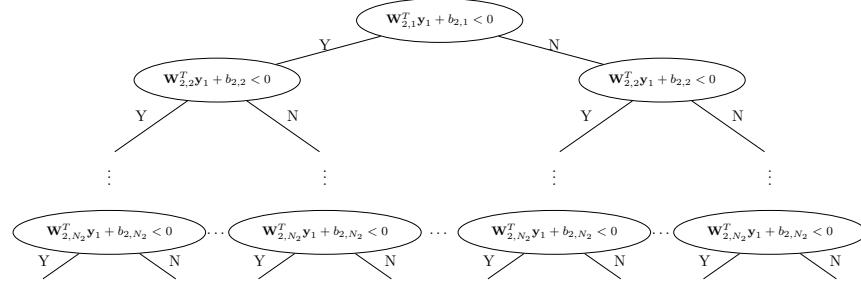


Figure 12.7: The resulting subtree $\mathcal{T}_{2,2}(\mathbf{y}_1)$ for $\mathbf{y}_1 = (0, \dots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})$. A, B, C are as follows:

- A is $\mathbf{W}_{2,1}^T(0, \dots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T + b_{2,1} < 0$.
- B is $\mathbf{W}_{2,2}^T(0, \dots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T + b_{2,2} < 0$.
- C is $\mathbf{W}_{2,N_2}^T(0, \dots, \mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1})^T + b_{2,N_2} < 0$.

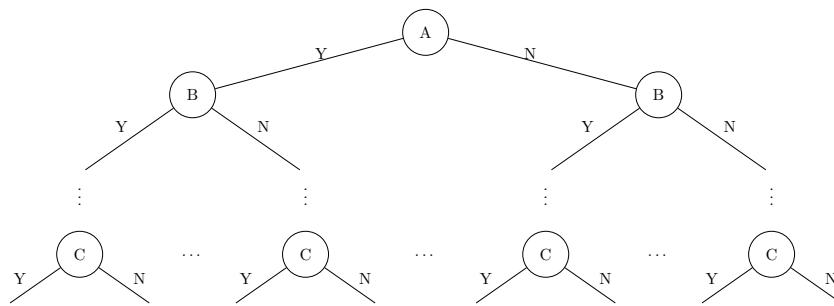
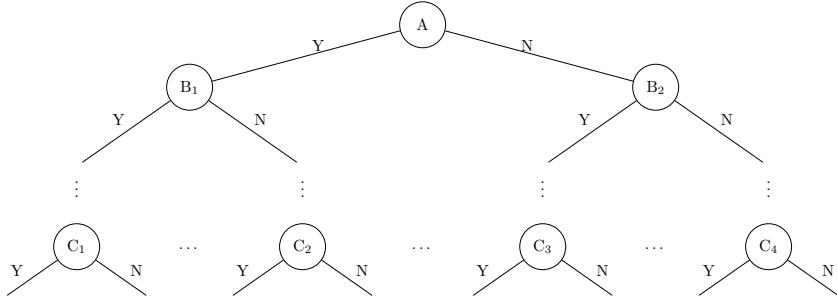


Figure 12.8: The subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$. The labels $A, B_1, B_2, C_1, C_2, C_3, C_4$ are as follows:

- A is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,2})^T \mathbf{y}_L + b_{O,1} - b_{O,2} < 0$.
- B_1 is $(\mathbf{W}_{O,2} - \mathbf{W}_{O,3})^T \mathbf{y}_L + b_{O,2} - b_{O,3} < 0$.
- B_2 is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,3})^T \mathbf{y}_L + b_{O,1} - b_{O,3} < 0$.
- C_1 is $(\mathbf{W}_{O,q-1} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,q-1} - b_{O,q} < 0$.
- C_2 is $(\mathbf{W}_{O,5} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,5} - b_{O,q} < 0$.
- C_3 is $(\mathbf{W}_{O,3} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,3} - b_{O,q} < 0$.
- C_4 is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,q})^T \mathbf{y}_L + b_{O,1} - b_{O,q} < 0$.



We model the output layer similarly by observing that in this layer we calculate

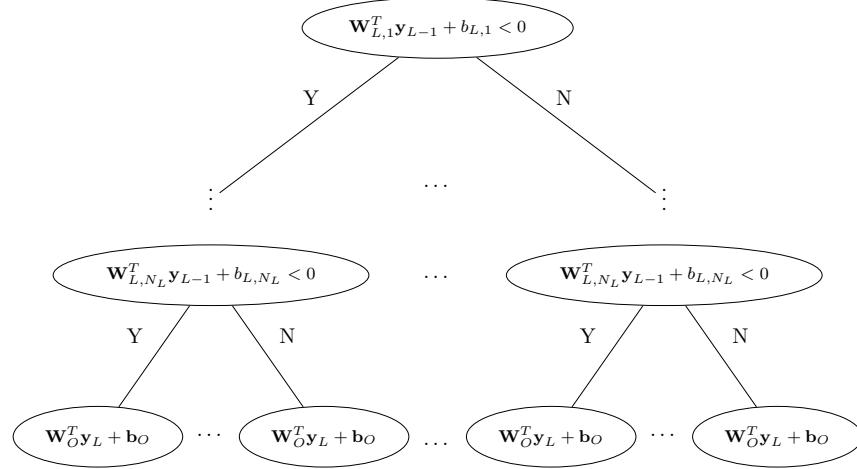
$$\text{argmax}_{i \in [q]} (\mathbf{W}_{O,i}^T \mathbf{y}_L + b_{O,i}) \quad (12.16)$$

in order to find the output class. The construction of the subtree $\mathcal{T}_{2,O}(\mathbf{y}_L)$ is depicted in Figure 12.8.

We simulate the calculation of (12.16) with $\mathcal{T}_{2,O}(\mathbf{y}_L)$ in the following manner. Node A checks whether $y_{O,1}$ or $y_{O,2}$ is larger. Node B_1 checks whether $y_{O,2}$ or $y_{O,3}$ is larger conditioned that $y_{O,2} > y_{O,1}$. Node B_2 checks whether $y_{O,1}$ or $y_{O,3}$ is larger conditioned that $y_{O,1} > y_{O,2}$, and so on. Each branch of the tree thus explicitly calculates which output node outputs the highest value (using a lexicographic decision rule in case of ties). We can then assign the class associated with that node as the output for the appropriate leaf nodes of $\mathcal{T}_{2,O}(\mathbf{y}_L)$. Since at each branch we know \mathbf{y}_L as an explicit function of \mathbf{x} , we also have that $\mathcal{T}_{2,O}(\mathbf{y}_L)$ is a decision tree where all inequalities are explicitly written linear functions of \mathbf{x} as well.

By the construction of \mathcal{T}_2 , the output value of \mathcal{T}_2 is the same as \mathcal{N}_2 given an input vector \mathbf{x} . Since \mathcal{T}_2 is a classification tree with hyperplanes, it follows that an OCT-H has at least as good accuracy as \mathcal{T}_2 , proving the theorem. \square

Figure 12.9: Subtree $\mathcal{T}_{2,L}(\mathbf{y}_{L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $\sum_{\ell=1}^L N_\ell$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as outputs.



ReLU Regression FNNs and ORT-Hs

If the network is a regression neural network \mathcal{N}_2 instead of a classification neural network, meaning it outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O \in \mathbb{R}^q$, we are also able to build a regression tree \mathcal{T}_2 to make the same predictions as the neural network. We build the same decision tree as in the proof of Theorem 12.2 in Section 12.2 by building subtrees up until subtree $\mathcal{T}_{2,L}(\mathbf{y}_{L-1})$, the subtree with splits based on weights and biases $\mathbf{W}_{L,i}, b_{L,i}, i \in [N_L]$. This results in a tree of depth $\sum_{\ell=1}^L N_\ell$. Then, for each leaf node of this tree we assign the linear function $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as the output value, where by the construction of the tree \mathbf{y}_L is a linear function of \mathbf{x} . Through this process, \mathcal{T}_2 calculates the same output as \mathcal{N}_2 given an input vector \mathbf{x} . Since \mathcal{T}_2 is a regression tree with hyperplane splits, it follows that ORT-H has at least as good accuracy as \mathcal{T}_2 , completing this extension of Theorem 12.2. An example of the final subtree in this example is depicted in Figure 12.9.

12.3 CNNs and Optimal Trees

In this section, we construct an OCT-H (ORT-H) that can classify training data at least as well as a classification (regression) convolutional neural network with perceptron or rectified linear unit activation functions.

Perceptron Classification CNNs and OCT-Hs

The key result in this section is as follows.

Theorem 12.3. *An OCT-H of depth N_1 can classify training data at least as well as a given classification CNN with the perceptron activation function, max pooling layers, and N_1 nodes in the first hidden layer.*

Proof. Our proof is constructive. We are given a CNN \mathcal{N}_3 with the following characteristics:

- The perceptron activation function as defined in (12.4).
- Output function $\phi_O(\mathbf{y}_O) : [0, 1]^q \rightarrow [0, 1]^q$.
- L hidden layers and one output layer, indexed $\ell \in [L]$.
- Max pooling layers as defined in (12.8).
- N_ℓ nodes in each hidden layer, indexed $i \in [N_\ell]$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.
- L_c pooling layers.
- R_ℓ nodes in pooling layer, indexed $i \in [R_\ell]$.
- Each node in a pooling layer takes as input a set $S_{\ell,i}$ of nodes from the previous convolutional layer with the properties defined in Section 12.1.
- q nodes in the output layer.

We construct a decision tree \mathcal{T}_3 with hyperplane splits and maximum depth N_1 that makes the same predictions as \mathcal{N}_3 . It follows that an OCT-H of maximum depth N_1 has at least the same classification accuracy as \mathcal{N}_3 .

Given the inequality from the first node in the first hidden layer of \mathcal{N}_3 defined by the weight vector $\mathbf{W}_{1,1}$ and bias scalar $b_{1,1}$, we define the first split of \mathcal{T}_3 as

$$\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1} < 0, \quad (12.17)$$

Figure 12.10 provides a visualization of this split.

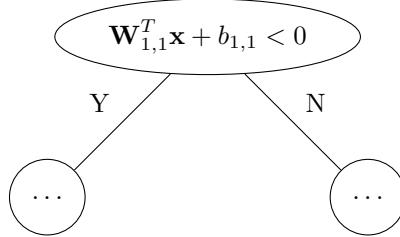
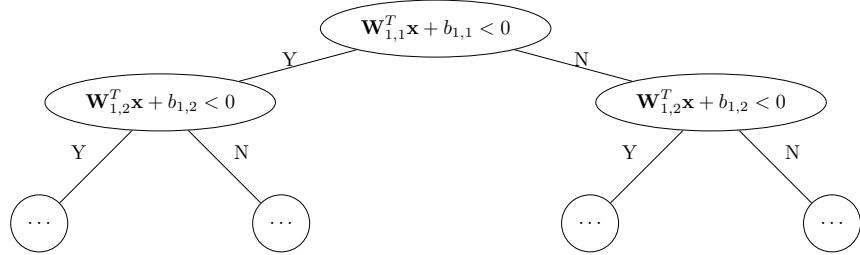
The second split is given by

$$\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2} < 0. \quad (12.18)$$

This new split is visualized in Figure 12.11.

We continue this process for all N_1 nodes in the first hidden layer, building a decision tree of depth N_1 with every split at depth N_1 being given by

$$\mathbf{W}_{1,N_1}^T \mathbf{x} + b_{1,N_1} < 0. \quad (12.19)$$

Figure 12.10: The first split of decision tree \mathcal{T}_3 .**Figure 12.11:** The decision tree we are building up to depth 2.

To complete the construction of \mathcal{T}_3 , we need to assign a classification value for every leaf of \mathcal{T}_3 . Given an input \mathbf{x} to \mathcal{N}_3 , there are 2^{N_1} binary vectors that could be output by the first hidden layer of \mathcal{N}_3 . These 2^{N_1} vectors, by our construction of \mathcal{T}_3 , exactly correspond to the 2^{N_1} leaves of \mathcal{T}_3 . Given the output $\mathbf{y}_1(\mathbf{x})$ of the first pooling layer, the final class prediction of \mathcal{N}_3 will be $k(\mathbf{y}_1)$, which follows deterministically given the \mathbf{y}_1 vector and the $\mathbf{W}_{\ell,i}$, $b_{\ell,i}$ values by using the calculation process outlined in Section 2.2. In every leaf node r of the tree we assign the classification value $k(\mathbf{y}_1)$. Thus, by our construction for every input \mathbf{x} , the output of \mathcal{T}_3 is the same as the output of the neural network \mathcal{N}_3 . Since the optimal tree must do at least as well as \mathcal{T}_3 , it must do at least as well as \mathcal{N}_3 , and the proof is complete. \square

Perceptron Regression CNNs and ORT-Hs

One can extend Theorem 12.3 in Section 12.3 to the case of a regression CNN, where $\phi_O(\mathbf{y}_L) \in \mathbb{R}^q$. To do this, note that because the first hidden layer can output at most 2^{N_1} unique vectors there are only 2^{N_1} possible output values of the network. In this case, one can modify the proof of Theorem 12.3 by assigning these 2^{N_1} unique values to the leaf nodes of the decision tree in the place of classification values. With this adjustment, extending the above proof to regression CNNs with perceptron activation functions is straightforward.

ReLU Classification CNNs and OCT-Hs

While the result in Section 4.1 follows directly from Theorem 12.1, the addition of pooling layers implies that the OCT-H construction given in Theorem 12.2 does not apply to CNNs with ReLU activation functions. The next theorem illustrates a different OCT-H construction for this case.

Theorem 12.4. *An OCT-H with depth $q-1+\sum_{\ell=1}^L N_\ell$ can classify training data at least as well as a given CNN with the rectified linear unit activation function (12.5), max pooling layers as defined in (12.8), and output function defined in Eq. (12.6).*

Proof. Our proof is constructive. We are given a CNN \mathcal{N}_4 as described in Section 12.1 with the following characteristics:

- The rectified linear unit activation function defined in (12.5).
- The output function as defined in (12.6).
- L hidden layers and one output layer, indexed $\ell \in [L]$.
- Max pooling layers as defined in (12.8).
- N_ℓ nodes in each hidden layer, indexed $i \in [N_\ell]$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.
- L_c pooling layers.
- R_ℓ nodes in pooling layer, indexed $i \in [R_\ell]$.
- Each node in a pooling layer takes as input a set $S_{\ell,i}$ of nodes from the previous convolutional layer with the properties defined in Section 12.1.
- q nodes in the output layer.

We now construct a decision tree \mathcal{T}_4 with hyperplane splits of maximum depth $q-1+\sum_{\ell=1}^L N_\ell$ that makes the same predictions as \mathcal{N}_4 . It follows that an OCT-H of maximum depth $q-1+\sum_{\ell=1}^L N_\ell$ has at least the same classification accuracy as \mathcal{N}_4 .

First, we construct a subtree $\mathcal{T}_{4,1}$ with splits based on the output of first node in the first pooling layer $P_{1,1}$ as defined in Eq. (12.8). Without loss of generality, assume that $S_{1,1} = \{1, \dots, k_1\}$. Then the first split of $\mathcal{T}_{4,1}$ is

$$(\mathbf{W}_{1,1} - \mathbf{W}_{1,2})^T \mathbf{x} + (b_{1,1} - b_{1,2}) < 0. \quad (12.20)$$

This split is depicted in Figure 12.12, and determines whether the maximum among the first two hyperplanes is $\mathbf{W}_{1,1}^T \mathbf{x} + b_{1,1}$ or $\mathbf{W}_{1,2}^T \mathbf{x} + b_{1,2}$.

Figure 12.12: The first split of classification (regression) tree $\mathcal{T}_{4,1}$.

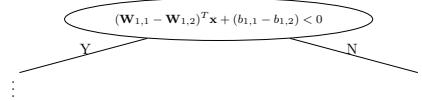
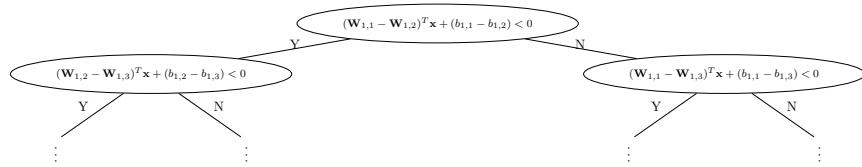


Figure 12.13: The decision tree $\mathcal{T}_{4,1}$ we are building up to depth 2. The identity of the hyperplane that achieves the maximum is from left to right: 3, 2, 3 and 1.



If Inequality (12.20) holds, then we next to check whether

$$(\mathbf{W}_{1,2} - \mathbf{W}_{1,3})^T \mathbf{x} + (b_{1,2} - b_{1,3}) < 0, \quad (12.21)$$

while if Inequality (12.20) does not hold, we need to check whether

$$(\mathbf{W}_{1,1} - \mathbf{W}_{1,3})^T \mathbf{x} + (b_{1,1} - b_{1,3}) < 0, \quad (12.22)$$

thus forming the hyperplane splits for $\mathcal{T}_{4,1}$ for depth 2 depicted in Figure 12.13.

At depth $k_1 - 1$ tree $\mathcal{T}_{4,1}$ will determine the identity of i^* :

$$i^* = \arg \max_{i \in [k_1]} (\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i}). \quad (12.23)$$

After the $(k_1 - 1)$ th hyperplane split, we append to $\mathcal{T}_{4,1}$ the final hyperplane split:

$$\mathbf{W}_{1,i^*}^T \mathbf{x} + b_{1,i^*} < 0, \quad (12.24)$$

at the appropriate node. The tree $\mathcal{T}_{4,1}$ is depicted in Figure 12.14.

The tree $\mathcal{T}_{4,1}$ has depth k_1 and the construction is complete.

Following this, we repeat this process for all nodes in the first hidden and pooling layers, building a tree of depth N_1 that can be used to calculate the output of the pooling layer \mathbf{P}_1 . In each of the branches at depth N_1 of this new subtree we implicitly calculate the corresponding \mathbf{P}_1 values. For example, the first branch corresponds to $(0, \dots, 0)^T$, the second corresponds to $(0, \dots, 0, \max_{i \in [k_1]} (\mathbf{W}_{1,i}^T \mathbf{x} + b_{1,i}))^T$, etc.

We then model the second hidden layer of \mathcal{N}_4 by constructing after each branch a new subtree of depth k_2 as in Figure 12.15, but using the corresponding value of \mathbf{P}_1 instead of \mathbf{x} in the splits. We call this subtree

Figure 12.14: The resulting tree $\mathcal{T}_{4,1}$. The labels A, B_1, \dots, C_4 and D_{i^*} are as follows:

- A is $(\mathbf{W}_{1,1} - \mathbf{W}_{1,2})^T \mathbf{x} + (b_{1,1} - b_{1,2}) < 0$
- B_1 is $(\mathbf{W}_{1,2} - \mathbf{W}_{1,3})^T \mathbf{x} + (b_{1,2} - b_{1,3}) < 0$
- B_2 is $(\mathbf{W}_{1,1} - \mathbf{W}_{1,3})^T \mathbf{x} + (b_{1,1} - b_{1,3}) < 0$
- C_1 is $(\mathbf{W}_{1,k_1-1} - \mathbf{W}_{1,k_1})^T \mathbf{x} + (b_{1,k_1-1} - b_{1,k_1}) < 0$
- C_2 is $(\mathbf{W}_{1,10} - \mathbf{W}_{1,k_1})^T \mathbf{x} + (b_{1,10} - b_{1,k_1}) < 0$
- C_3 is $(\mathbf{W}_{1,3} - \mathbf{W}_{1,k_1})^T \mathbf{x} + (b_{1,3} - b_{1,k_1}) < 0$
- C_4 is $(\mathbf{W}_{1,1} - \mathbf{W}_{1,k_1})^T \mathbf{x} + (b_{1,1} - b_{1,k_1}) < 0$
- D_{i^*} is $\mathbf{W}_{1,i^*}^T \mathbf{x} + b_{1,i^*} < 0$.

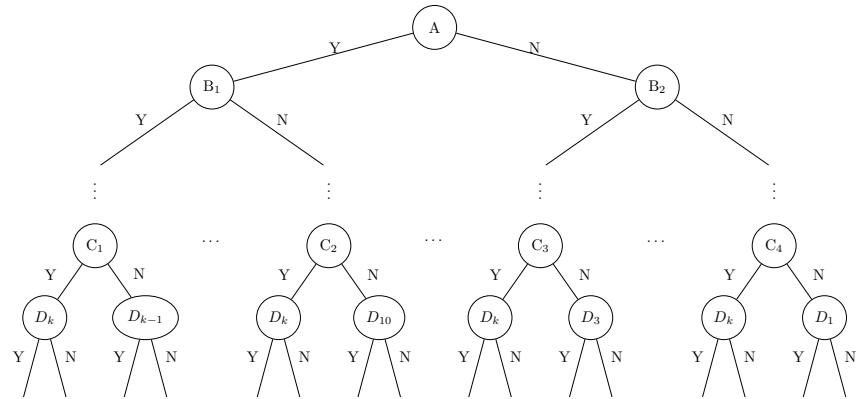
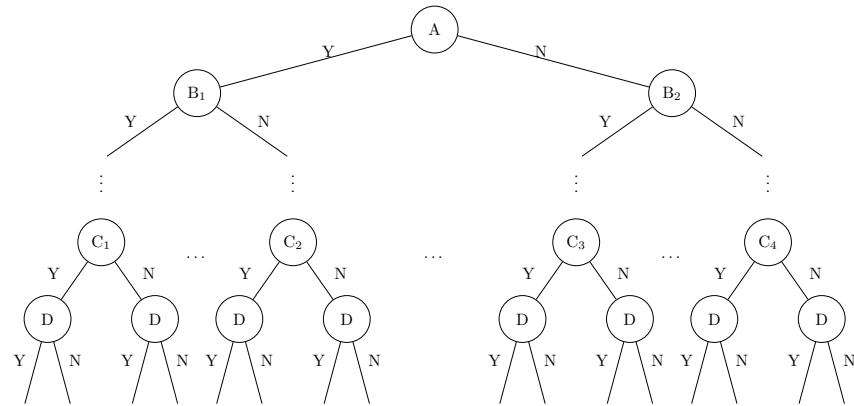


Figure 12.15: The subtree $\mathcal{T}_{4,2}(\mathbf{P}_1)$ up to depth k_2 . The labels A, B₁, ..., D are as follows:

- A is $(\mathbf{W}_{2,1} - \mathbf{W}_{2,2})^T \mathbf{P}_1 + (b_{2,1} - b_{2,2}) < 0$
- B₁ is $(\mathbf{W}_{2,2} - \mathbf{W}_{2,3})^T \mathbf{P}_1 + (b_{2,2} - b_{2,3}) < 0$
- B₂ is $(\mathbf{W}_{2,1} - \mathbf{W}_{2,3})^T \mathbf{P}_1 + (b_{2,1} - b_{2,3}) < 0$
- C₁ is $(\mathbf{W}_{2,k_2-1} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,k_2-1} - b_{2,k_2}) < 0$
- C₂ is $(\mathbf{W}_{2,10} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,10} - b_{2,k_2}) < 0$
- C₃ is $(\mathbf{W}_{2,3} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,3} - b_{2,k_2}) < 0$
- C₄ is $(\mathbf{W}_{2,1} - \mathbf{W}_{2,k_2})^T \mathbf{P}_1 + (b_{2,1} - b_{2,k_2}) < 0$
- D is $\mathbf{W}_{2,i^*}^T \mathbf{P}_1 + b_{2,i^*} < 0$



$\mathcal{T}_{4,2}(\mathbf{P}_1)$, as it is the first of the subtrees created based on the neural network hidden layer two weights and the \mathbf{P}_1 .

We use the same process for adapting the remaining convolutional-pooling layer pairs. After we have finished modeling those pairs of layers, the process of modeling the remaining hidden layers and the output layer nodes as a decision tree is exactly the same as it was in the proof of Theorem 2. This once again results in the construction of a tree with splits based deterministically on \mathbf{x} .

By the construction of \mathcal{T}_4 , \mathcal{T}_4 calculates the same output as \mathcal{N}_4 given an input vector \mathbf{x} . Since \mathcal{T}_4 is a decision tree with hyperplane splits, it follows that an optimal tree has at least as good accuracy as \mathcal{T}_4 , proving the theorem. \square

ReLU Regression CNNs and ORT-Hs

If the given CNN is a regression neural network \mathcal{N}_4 instead of a classification neural network, meaning it outputs $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O \in \mathbb{R}^q$, we are also able to build a regression tree \mathcal{T}_4 to make the same predictions as the neural network. We build the same decision tree as in the proof of Theorem 12.4 in Section 12.3 by building subtrees up until subtree $\mathcal{T}_{4,L}(\mathbf{y}_{L-1})$, the subtree with splits based on weights and biases $\mathbf{W}_{L,i}, b_{L,i}, i \in [N_L]$. This results in a tree of depth $\sum_{\ell=1}^L N_\ell$. Then, for each leaf node of this tree we assign the linear function $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as the output value, where by the construction of the tree \mathbf{y}_L is a linear function of \mathbf{x} . Through this process, \mathcal{T}_4 calculates the same output as \mathcal{N}_4 given an input vector \mathbf{x} . Since \mathcal{T}_4 is a regression tree with hyperplane splits, it follows that ORT-H has at least as good accuracy as \mathcal{T}_4 , completing this extension of Theorem 4. An example of the final subtree in this example is depicted in Figure 12.16.

12.4 RNNs and Optimal Trees

In this section, we construct an OCT-H (ORT-H) that can be used to classify training data at least as well as a classification (regression) recurrent neural network with perceptron or rectified linear unit activation functions.

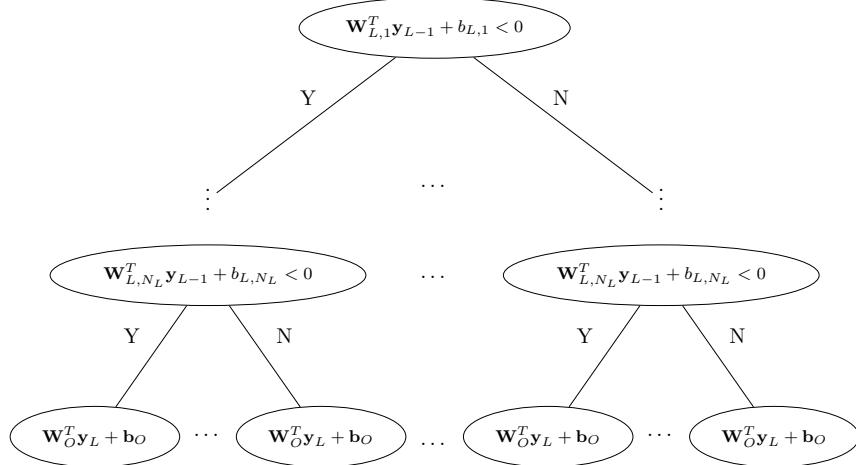
Perceptron Classification RNNs and OCT-Hs

The key result in this section is as follows:

Theorem 12.5. *An OCT-H with maximum depth $T^* \times N_1$ can classify sequential data with T^* terms per sequence in a training set at least as well as a given classification RNN with the perceptron activation function, one hidden layer containing N_1 nodes, and q nodes in the output layer.*

Proof. Our proof is constructive. We are given a classification RNN \mathcal{N}_5 as described in Section 12.1 with the following specifications:

Figure 12.16: Subtree $\mathcal{T}_{4,L}(\mathbf{y}_{L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $\sum_{\ell=1}^L N_\ell$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_L + \mathbf{b}_O$ as outputs.



- The perceptron activation function as defined in (12.4).
- Output function $\phi_O(\mathbf{y}_O) : [0, 1]^q \rightarrow [0, 1]^q$.
- One hidden layer and one output layer.
- N_1 nodes in the hidden layer, indexed $i \in [N_1]$.
- q nodes in the output layer, indexed $i \in [q]$.
- Node $n_{1,i}$ characterized by $\mathbf{W}_{g,i}, \mathbf{W}_{h,i}, b_{1,i}$.

Using \mathcal{N}_5 , we construct a decision tree \mathcal{T}_5 with hyperplanes and maximum depth $T^* \times N_1$ that can be used to make the same predictions as the network. It follows that an OCT-H of maximum depth $T^* \times N_1$ has at least the same classification accuracy as \mathcal{N}_5 .

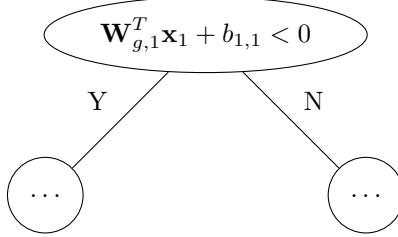
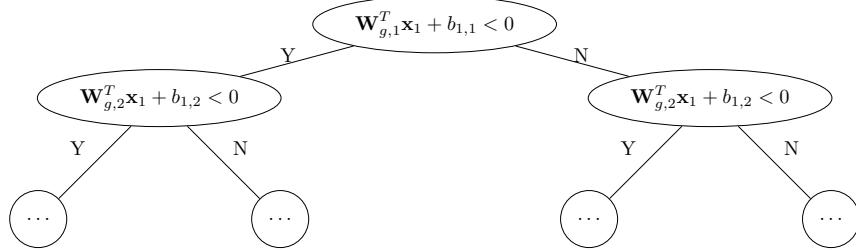
First, for ease of notation we define \mathbf{x}^* as

$$\mathbf{x}^* = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{T^*})^T, \quad (12.25)$$

which is the concatenation of all the vectors in the input sequence $(\mathbf{x}_t)_{t \in [T^*]}$. Then we define the first split of \mathcal{T}_5 as

$$(\mathbf{W}_{g,1}, \mathbf{0}, \dots, \mathbf{0}) \mathbf{x}^* + b_{1,1} = \mathbf{W}_{g,1}^T \mathbf{x}_1 + b_{1,1} < 0, \quad (12.26)$$

where $(\mathbf{W}_{g,1}, \mathbf{0}, \dots, \mathbf{0})$ is the concatenation of $\mathbf{W}_{g,1}$ horizontally with $T^* - 1$ zero vectors $\mathbf{0}$ with the same dimensions as $\mathbf{W}_{g,1}$. This results in the simple split seen in Figure 12.17.

Figure 12.17: The first split of decision tree \mathcal{T}_5 .**Figure 12.18:** The first two depths of tree \mathcal{T}_5 .

Independent of whether inequality (12.26) is satisfied or not, the second split is given by

$$(\mathbf{W}_{g,2}, \mathbf{0}, \dots, \mathbf{0})\mathbf{x}^* + b_{1,2} = \mathbf{W}_{1,2}^T \mathbf{x}_1 + b_{1,2} < 0, \quad (12.27)$$

where $(\mathbf{W}_{g,2}, \mathbf{0}, \dots, \mathbf{0})$ is the concatenation of $\mathbf{W}_{g,2}$ horizontally with $T^* - 1$ zero vectors $\mathbf{0}$ with the same dimensions as $\mathbf{W}_{g,2}$. Figure 12.18 provides a visualization of the new branches we added to the tree in Figure 12.17.

We continue this process for all N_1 nodes in the first hidden layer, building a decision tree of depth N_1 , with every split at depth N_1 being given by

$$\mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1} < 0. \quad (12.28)$$

The resultant subtree is shown in Figure 12.19

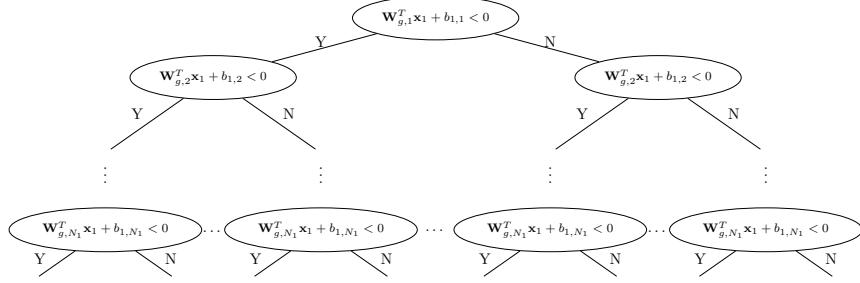
This is the subtree that simulates the output of the hidden layer \mathcal{N}_5 after the first time step.

After depth N_1 , there are 2^{N_1} branches, as shown in Figure 12.19. Note that there are 2^{N_1} possible output vectors $\mathbf{y}_{1,1}$ of the first layer of \mathcal{N}_5 ,

$$(0, \dots, 0)^T, (1, \dots, 0)^T, \dots, (1, \dots, 1)^T.$$

These 2^{N_1} possible values of $\mathbf{y}_{1,1}$ correspond to the 2^{N_1} branches in the tree \mathcal{T}_5 shown in Figure 12.19. In each of these branches we have implicitly calculated the corresponding $\mathbf{y}_{1,1}$ values. For example, the first branch corresponds to $(0, \dots, 0)^T$, the second corresponds to $(0, \dots, 0, 1)^T$, etc.

Figure 12.19: The decision tree \mathcal{T}_5 we are building up to depth N_1 .



We then model \mathcal{N}_5 at the second time step by constructing after each branch a new subtree of depth N_1 as in Figure 12.20, but with the corresponding value of $\mathbf{y}_{1,1}$ being used as a constant value in addition to \mathbf{x}_2 . Thus, the first split of this new subtree is

$$(\mathbf{0}, \mathbf{W}_{g,1}, \mathbf{0}, \dots, \mathbf{0})\mathbf{x}^* + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} = \mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} < 0. \quad (12.29)$$

This process continues for the remaining nodes of the first hidden layer shown in Figure 12.20, resulting in the subtree shown in Figure 12.20.

In the subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ in Figure 12.20 we substitute in the corresponding binary vector $\mathbf{y}_{1,1}$. For example, if $\mathbf{y}_{1,1} = (0, \dots, 0, 1)^T$, then subtree $\mathcal{T}_{5,2}((0, \dots, 0, 1)^T)$ is depicted in Figure 12.21.

Given that at each branch we know exactly $\mathbf{y}_{1,1}$, $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ is a decision tree where all inequalities are explicitly written as linear functions of \mathbf{x}^* .

Continuing in this way we model the output vector $\mathbf{y}_{t,1}$ of the t th time step of \mathcal{N}_5 as a classification tree by defining $\mathbf{y}_{t,1}$ based on the path taken down the previous subtree. At the t th subtree, only the weights of the \mathbf{x}_t vector contained in the vector \mathbf{x}^* are non-zero.

To complete the construction of \mathcal{T}_5 , we need to assign a classification value for every leaf of \mathcal{T}_5 . At time step T^* , there are 2^{N_1} possible binary vectors that the first hidden layer of \mathcal{N}_5 could output. These 2^{N_1} vectors, by our construction of \mathcal{T}_5 , exactly correspond to the 2^{N_1} leaves of the final subtree of \mathcal{T}_5 . Given the output $\mathbf{y}_{T^*,1}(\mathbf{x}_{T^*})$ of the hidden layer, the final prediction of \mathcal{N}_5 will be $k(\mathbf{y}_{T^*,1})$, which is calculated deterministically given the $\mathbf{y}_{T^*,1}$ vector and the $\mathbf{W}_{O,i}$, $b_{O,i}$ values by using the process outlined in Section 12.1. In every node r of the tree we assign the classification value $k(\mathbf{y}_{T^*,1})$.

We next show that the output of \mathcal{T}_5 is the same as the output of \mathcal{N}_5 for input data sequence \mathbf{x}_t . To see this, if \mathbf{x}_t is input into \mathcal{N}_5 , the hidden layer outputs $\mathbf{y}_{1,1}$ at the first time step, $\mathbf{y}_{2,1}$ at the second time step, and so on, until at last the sequence is assigned classification value $k(\mathbf{y}_{T^*,1})$.

Figure 12.20: Subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ of depth N_1 is concatenated to the corresponding branch of the subtree depicted in Figure 12.19, resulting in a subtree of depth $2N_1$. The labels of A, B, C are as follows:

- A is $\mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T \mathbf{x}_2 + \mathbf{W}_{h,2}^T \mathbf{y}_{1,1} + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T \mathbf{x}_2 + \mathbf{W}_{h,N_1}^T \mathbf{y}_{1,1} + b_{1,N_1} < 0$.

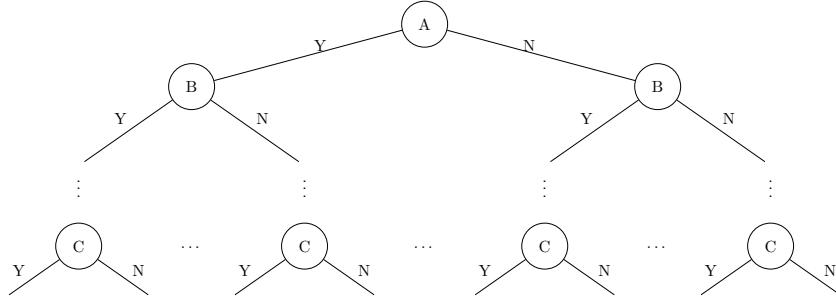
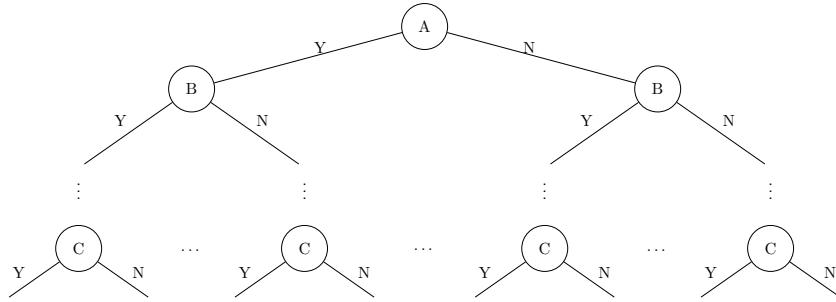


Figure 12.21: The resulting subtree $\mathcal{T}_{5,2}(\mathbf{y}_{1,1})$ for $\mathbf{y}_{1,1} = [0, \dots, 0, 1]$. The labels of A, B, C are as follows:

- A is $\mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T (0, \dots, 0, 1)^T + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T \mathbf{x}_2 + \mathbf{W}_{h,2}^T (0, \dots, 0, 1)^T + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T \mathbf{x}_2 + \mathbf{W}_{h,N_1}^T (0, \dots, 0, 1)^T + b_{1,N_1} < 0$.



However, by the construction of the tree, the point \mathbf{x}^* is sorted down the paths corresponding to $\mathbf{y}_{1,1}$, $\mathbf{y}_{2,1}$, and so on, until it is sorted to the leaf node corresponding to $k(\mathbf{y}_{T^*,1})$ by construction. Thus, for a given data sequence \mathbf{x}_t , the network and the tree predict the same classification value.

Since an OCT-H does at least as well as \mathcal{T}_5 in classifying the training data, it must do at least as well as \mathcal{N}_5 too. Thus, by construction, we have that an optimal decision tree with maximum depth $T^* \times N_1$ can classify data in a training set at least as well as the given FNN with perceptron activation function and one hidden layer with N_1 nodes in it, completing the proof of the theorem. \square

Perceptron Regression RNNs and ORT-Hs

In the case where we have a regression RNN with the perceptron function, meaning $\phi_O(\mathbf{y}_L) \in \mathbb{R}^q$, then because the first hidden layer can output at most 2^{N_1} unique vectors there are only 2^{N_1} possible output values of the network. In this case, one can modify the proof of Theorem 12.5 in Section 12.4 by assigning these 2^{N_1} unique values to the leaf nodes of the decision tree in the place of classification values. With this adjustment, extending the above proof to regression RNNs with perceptron activation functions is straightforward.

ReLU Classification RNNs and OCT-Hs

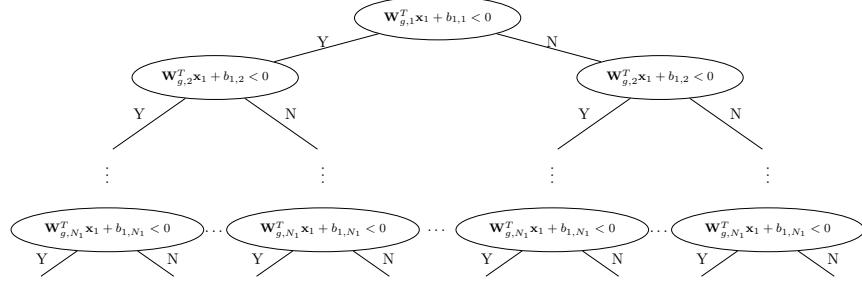
In this section, we show that given a RNN with rectified linear unit activation functions, we can construct an ORT-H that can be used to classify given training data at least as well as that network. The theorem is as follows:

Theorem 12.6. *An OCT-H with maximum depth $T^* \times N_1 + q - 1$ can classify sequential data with T^* terms per sequence in a training set at least as well as a given classification RNN with the ReLU activation function, one hidden layer containing N_1 nodes, and q nodes in the output layer.*

Proof. Our proof is constructive. Given that we have a recurrent neural network \mathcal{N}_6 with:

- The ReLU activation function;
- One hidden layer and one output layer;
- N_1 nodes in the hidden layer, indexed $i \in [N_1]$;
- q nodes in the output layer, indexed $i \in [q]$;
- Hidden layer node $n_{1,i}$ defined by $\mathbf{W}_{g,i}, \mathbf{W}_{h,i}, b_{1,i}$;

Figure 12.22: The decision tree \mathcal{T}_6 we are building up to depth N_1 .



we construct a decision tree \mathcal{T}_6 with hyperplanes and maximum depth $T^* \times N_1 + q - 1$ that makes the same predictions as \mathcal{N}_6 . It follows that an optimal tree with hyperplanes of maximum depth $T^* \times N_1 + q - 1$ has at least the same classification accuracy as \mathcal{N}_6 .

First, for ease of notation we define \mathbf{x}^* as we did in Eq. (12.25). Then we build the tree up to depth N_1 exactly as we did in the proof in Section 12.4. This part of the tree is shown in Figure 12.22.

After depth N_1 , there are 2^{N_1} branches, as shown in Figure 12.22. Note that there are 2^{N_1} possible output vectors $\mathbf{y}_{1,1}$ of the hidden layer of \mathcal{N}_6 at time step 1,

$$(0, \dots, 0)^T, (\mathbf{W}_{g,1}^T \mathbf{x}_1 + b_{1,1}, \dots, 0)^T, \dots, (\mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1}, \dots, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T,$$

as in each node of the first hidden layer \mathcal{N}_6 computes

$$(\mathbf{y}_{1,1})_i = \max\{\mathbf{W}_{g,i}^T \mathbf{x}_1 + b_{1,i}, 0\}, \quad i \in [N_1] \quad (12.30)$$

at time step 1.

These 2^{N_1} possible values of $\mathbf{y}_{1,1}$ correspond to the 2^{N_1} branches in the tree \mathcal{T}_6 shown in Figure 12.22. In each of these branches we have implicitly calculated the corresponding $\mathbf{y}_{1,1}$ values. For example, the first branch corresponds to $(0, \dots, 0)^T$, the second corresponds to $(0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T$, etc.

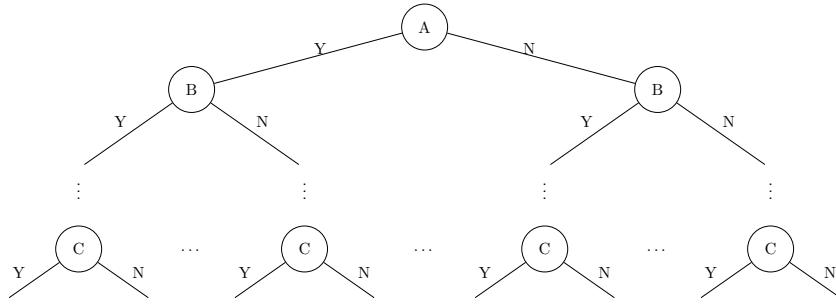
We then model \mathcal{N}_6 at the second time step by constructing after each branch a new subtree of depth N_1 as in Figure 12.20, but with the corresponding value of $\mathbf{y}_{1,1}$ being used as a constant value in addition to \mathbf{x}_2 . Thus, the first split of this new subtree is

$$(\mathbf{0}, \mathbf{W}_{g,1}, \dots, \mathbf{0})^T \mathbf{x}^* + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} = \mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} < 0. \quad (12.31)$$

This process continues for the remaining nodes of the first hidden layer are shown in Figure 12.23, resulting in the subtree shown in Figure 12.23.

Figure 12.23: Subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ of depth N_1 is concatenated to the corresponding branch of the subtree depicted in Figure 12.22, resulting in a subtree of depth $2N_1$. The labels of A, B, C are as follows:

- A is $\mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T \mathbf{y}_{1,1} + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T \mathbf{x}_2 + \mathbf{W}_{h,2}^T \mathbf{y}_{1,1} + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T \mathbf{x}_2 + \mathbf{W}_{h,N_1}^T \mathbf{y}_{1,1} + b_{1,N_1} < 0$.



In the subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ in Figure 12.23 we substitute in the corresponding binary vector $\mathbf{y}_{1,1}$. For example, if $\mathbf{y}_{1,1} = (0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T$, then subtree $\mathcal{T}_{6,2}((0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T)$ is depicted in Figure 12.24.

Given that at each branch we know exactly $\mathbf{y}_{1,1}$, $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ is a decision tree where all inequalities are explicitly written as linear functions of \mathbf{x}^* .

Continuing in this way we model the output vector $\mathbf{y}_{t,1}$ of the t th time step of \mathcal{N}_6 as a classification tree by propagating the values of $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_L$ as explicit linear functions of \mathbf{x}^* .

Following this, we model the output layer of the network the same way we did in Section 3.2, resulting in the subtree seen in Figure 12.25.

Given an output $\mathbf{y}_{T^*,1}$ of the final hidden layer of \mathcal{N}_6 , \mathcal{N}_6 calculates the output vector

$$\mathbf{y}_O = \mathbf{W}_O^T \mathbf{y}_{T^*,1} + \mathbf{b}_O, \quad (12.32)$$

and then $k = \arg \max(y_{O,i})$, outputting k as the classification prediction for input \mathbf{x}^* . We simulate this calculation with a subtree $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$ depicted in Figure 12.25. Node A checks whether $y_{O,1}$ or $y_{O,2}$ is larger. Node B₁ checks whether $y_{O,2}$ or $y_{O,3}$ is larger conditioned that $y_{O,2} > y_{O,1}$. Node B₂ checks whether $y_{O,1}$ or $y_{O,3}$ is larger conditioned that $y_{O,1} > y_{O,2}$, and so on. Each branch of the tree thus explicitly calculates which output node outputs the highest value (using a lexicographic decision rule in case of ties), and we can then assign the class associated with that node as the output to the appropriate leaf nodes of $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$. Since at each branch

fix reference

Figure 12.24: The resulting subtree $\mathcal{T}_{6,2}(\mathbf{y}_{1,1})$ for $\mathbf{y}_{1,1} = (0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T$. The labels of A, B, C are as follows:

- A is $\mathbf{W}_{g,1}^T \mathbf{x}_2 + \mathbf{W}_{h,1}^T (0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T + b_{1,1} < 0$.
- B is $\mathbf{W}_{g,2}^T \mathbf{x}_2 + \mathbf{W}_{h,2}^T (0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T + b_{1,2} < 0$.
- C is $\mathbf{W}_{g,N_1}^T \mathbf{x}_2 + \mathbf{W}_{h,N_1}^T (0, \dots, 0, \mathbf{W}_{g,N_1}^T \mathbf{x}_1 + b_{1,N_1})^T + b_{1,N_1} < 0$.

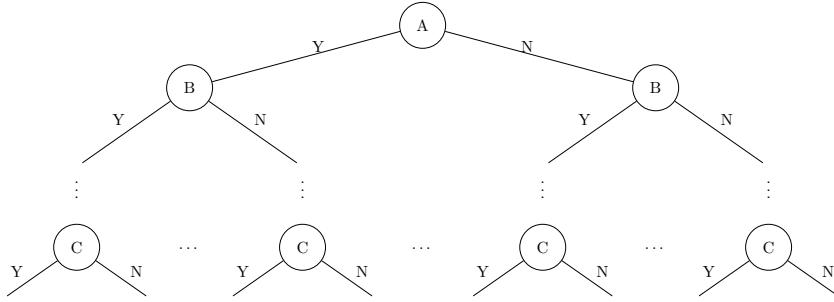
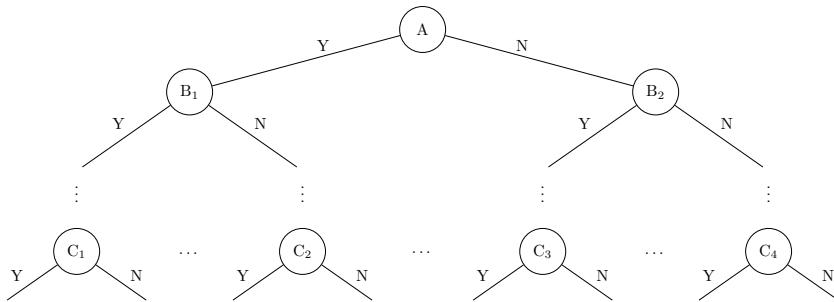


Figure 12.25: The subtree $\mathcal{T}_{6,O}(\mathbf{y}_1)$. The labels A, B₁, ..., C₄ are as follows:

- A is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,2})^T \mathbf{y}_{T^*,1} + b_{O,q} - b_{O,2}$
- B₁ is $(\mathbf{W}_{O,2} - \mathbf{W}_{O,3})^T \mathbf{y}_{T^*,1} + b_{O,2} - b_{O,3}$
- B₂ is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,3})^T \mathbf{y}_{T^*,1} + b_{O,1} - b_{O,3}$
- C₁ is $(\mathbf{W}_{O,q-1} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,q-1} - b_{O,q}$
- C₂ is $(\mathbf{W}_{O,5} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,5} - b_{O,q}$
- C₃ is $(\mathbf{W}_{O,3} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,3} - b_{O,q}$
- C₄ is $(\mathbf{W}_{O,1} - \mathbf{W}_{O,q})^T \mathbf{y}_{T^*,1} + b_{O,1} - b_{O,q}$



we know $\mathbf{y}_{T^*,1}$ as an explicit function of \mathbf{x} , we have that $\mathcal{T}_{6,O}(\mathbf{y}_{T^*,1})$ is a decision tree where all inequalities are explicitly written linear functions of \mathbf{x}^* as well.

We next show that \mathcal{T}_6 can be used to make the same predictions as \mathcal{N}_6 for input data sequence \mathbf{x}_t , $t \in [T^*]$. To see this, if \mathbf{x}_t is input into \mathcal{N}_6 , the hidden layer outputs $\mathbf{y}_{1,1}$ at the first time step, $\mathbf{y}_{2,1}$ at the second time step, and so on, until at last the sequence is assigned classification value $k(\mathbf{y}_{T^*,1})$. However, by the construction of the tree, the point \mathbf{x}^* is sorted down the paths corresponding to $\mathbf{y}_{1,1}$, $\mathbf{y}_{2,1}$, and so on, until it is sorted to the leaf node corresponding to $k(\mathbf{y}_{T^*,1})$ by construction. Thus, for a given data sequence \mathbf{x}_t , the network and the tree predict the same classification value.

Since an optimal tree does at least as well as \mathcal{T}_6 , that means that it must do at least as well as \mathcal{N}_6 too. Thus, by construction, we have that an optimal decision tree with maximum depth $T^* \times N_1 + q - 1$ that can be used to classify data in a training set at least as well as a given neural network with the perceptron activation function, 1 hidden layer, and N_1 nodes in the first hidden layer, completing the proof of the theorem. \square

ReLU Regression RNNs and ORT-Hs

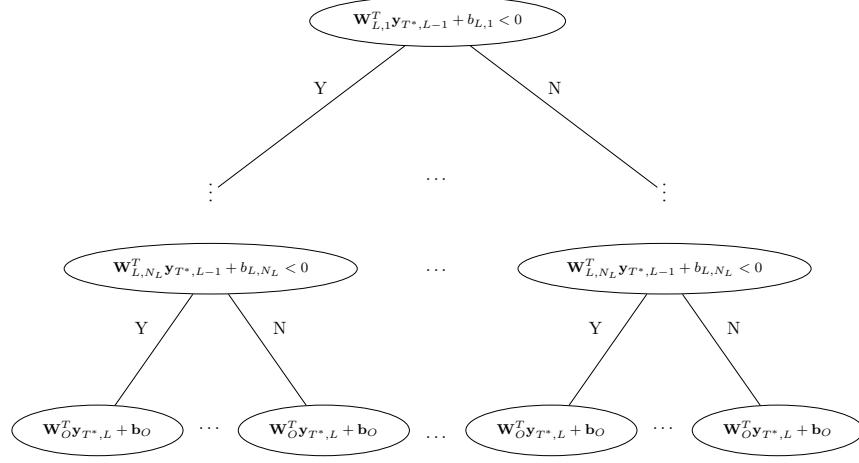
Note that if the network is a regression neural network \mathcal{N}_6 instead of a classification neural network, meaning it outputs $\mathbf{W}_O^T \mathbf{y}_{T^*,L} + \mathbf{b}_O \in \mathbb{R}^q$, we are also able to build a regression tree \mathcal{T}_6 to make the same predictions as the neural network. We build the same decision tree as in the proof of Theorem 12.6 in Section 12.4 by building subtrees up until subtree $\mathcal{T}_{6,L}(\mathbf{y}_{T^*,L-1})$, the subtree with splits based on weights and biases $\mathbf{W}_{L,i}, b_{L,i}$, $i \in [N_L]$. This results in a tree of depth $T^* \times N_1$. Then, for each leaf node of this tree we assign the linear function $\mathbf{W}_O^T \mathbf{y}_{T^*,L} + \mathbf{b}_O$ as the output value, where by the construction of the tree $\mathbf{y}_{T^*,L}$ is a linear function of \mathbf{x}^* . Through this process, \mathcal{T}_6 calculates the same output as \mathcal{N}_6 given an input \mathbf{x}^* . Since \mathcal{T}_6 is a regression tree with hyperplanesplits, it follows that ORT-H has at least as good accuracy as \mathcal{T}_6 , completing this extension of Theorem 6. An example of the final subtree in this example is depicted in Figure 12.26.

12.5 Transforming OCT-Hs into Classification NNs

In this section, we prove that given an OCT-H, there exists a neural network with perceptron activation functions that can classify the given training data at least as well as it. The theorem is as follows:

Theorem 12.7. *A neural network with perceptron activation functions (12.4), two hidden layers, N_1 nodes in the first hidden layer, and N_2 nodes in the*

Figure 12.26: Subtree $\mathcal{T}_{6,L}(\mathbf{y}_{T^*,L-1})$ is the last subtree built when we create the regression subtree. When concatenated onto the rest of the tree, we have built a tree of depth $T^* \times N_1$. Note that the leaves have linear functions $\mathbf{W}_O^T \mathbf{y}_{T^*,L} + \mathbf{b}_O$ as outputs.



second can classify training data at least as well as a given decision tree with N_1 split nodes and N_2 leaf nodes.

Proof. Our proof is constructive. We are given a decision tree \mathcal{T}_7 with

- Hyperplane splits.
- N_1 split nodes.
- Split node i characterized by split $\mathbf{w}_i^T \mathbf{x} + b_i < 0$, indexed $i \in [N_1]$.
- N_2 leaf nodes.
- Leaf node i characterized by classification value k , $k \in [q]$.

We show that we can find a neural network \mathcal{N}_7 with

- The perceptron activation function (12.4).
- Two hidden layers and one output layer, indexed $\ell = 1, 2, O$.
- N_ℓ nodes in each hidden layer, indexed $i \in [N_\ell]$.
- q nodes in the output layer, indexed $i \in [q]$.
- Node $n_{\ell,i}$ defined by $\mathbf{W}_{\ell,i}, b_{\ell,i}$.

that classifies data at least as well as the tree. We define the first hidden layer weights and biases of \mathcal{N}_7 as

$$\mathbf{W}_{1,i} \triangleq \mathbf{w}_i \text{ for } i \in [N_1], \quad (12.33)$$

$$b_{1,i} \triangleq b_i \text{ for } i \in [N_1]. \quad (12.34)$$

By these definitions, a node in the first hidden layer activates if and only if an input \mathbf{x} would take the right branch if it was sorted to the node with the same weights in \mathcal{T}_7 .

To construct the weights for the second hidden layer, first say that \mathbf{x} is assigned to the r th leaf node of \mathcal{T}_7 , $r \in [N_2]$. We then define

$$\mathcal{S}_{1,r} = \{i \mid \mathbf{w}_i^\top \mathbf{x} + b_i \geq 0\}, \quad (12.35)$$

which is the set of all nodes i of tree \mathcal{T}_7 where the inequality $\mathbf{w}_i^\top \mathbf{x} + b_i \geq 0$ is satisfied by an input \mathbf{x} sorted to the r th leaf node. Likewise define $\mathcal{S}_{2,r}$ as

$$\mathcal{S}_{2,r} = \{i \mid \mathbf{w}_i^\top \mathbf{x} + b_i < 0\}. \quad (12.36)$$

We then define $N_r = |\mathcal{S}_{1,r}|$, the number of elements in $\mathcal{S}_{1,r}$. For each leaf node r , it either has $N_r \geq 1$, or $N_r = 0$ — call the set of $r \in \{1, \dots, N_2\}$ such that $N_r \geq 1$

$$C^* = \{r \mid N_r \geq 1\}. \quad (12.37)$$

We define $\mathbf{W}_{2,r}$ and $\mathbf{b}_{2,r}$ differently in each of the two cases. For the first case, where $r \in C^*$, define for each $i \in [N_1]$:

$$(W_{2,r})_i = \begin{cases} \frac{1}{N_r}, & \text{if } i \in \mathcal{S}_{1,r}, \\ -2, & \text{if } i \in \mathcal{S}_{2,r}, \\ 0, & \text{otherwise,} \end{cases} \quad (12.38)$$

$$b_{2,r} = -\frac{N_r - 1}{N_r} - \frac{1}{N_r + 1}, \quad (12.39)$$

In the second case, for $i \in [N_1]$. and for $r \notin C^*$, we define

$$(W_{2,r})_i = \begin{cases} -1, & \text{if } i \in \mathcal{S}_{2,r}, \\ 0, & \text{otherwise,} \end{cases} \quad (12.40)$$

$$b_{2,r} = 0.1. \quad (12.41)$$

This completes the characterization of all the nodes in second hidden layer of \mathcal{N}_7 .

The N_2 nodes in the second hidden layer of \mathcal{N}_7 have a one-to-one correspondence with the leaf nodes of \mathcal{T}_7 . The weight definitions in equations (12.38) through (12.41) are designed to ensure that a node in the second hidden layer activates if and only if the input to \mathcal{N}_7 would have been

sorted to the corresponding leaf node in \mathcal{T}_7 . This means that the output of this hidden layer, $\mathbf{y}_2(\mathbf{x})$, is a vector in $[0, 1]^{N_2}$ with exactly one entry that is 1, and all others zero.

We prove this correspondence between the second hidden layer nodes of \mathcal{N}_7 and the leaf nodes of \mathcal{T}_7 for the two cases where $r \in C^*$ and $r \notin C^*$. In the first case, where $r \in C^*$, note that if \mathbf{x} was assigned to leaf node r we must have $y_{1,i} = 1$ for all $i \in \mathcal{I}_{1,r}$. This corresponds to \mathbf{x} taking the right branch at split node i . Otherwise, we know that the input vector would take a left branch at a node in the decision tree where it would need to take a right branch to arrive at leaf node r , and so it cannot have been assigned to r . Thus, if $y_{1,i} = 0$ for any $i \in \mathcal{I}_{1,r}$, the maximum possible value of

$$\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \quad (12.42)$$

is

$$\frac{N_r - 1}{N_r} - \frac{N_r - 1}{N_r} - \frac{1}{N_r + 1} < 0, \quad (12.43)$$

based on the weights we defined in Eqs. (12.38) and (12.39). This means we correctly have

$$\mathbb{1}\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\} = 0 \quad (12.44)$$

in this case.

Likewise, we need a point sorted to leaf node $r \in C^*$ in \mathcal{T}_7 to have $y_{1,i} = 0$ for every first hidden layer node $i \in \mathcal{I}_{2,r}$ of \mathcal{N}_7 , as by definition we must have had $\mathbf{w}_i^T \mathbf{x} + b_i < 0$ for \mathbf{x} to be assigned to r . This corresponds to \mathbf{x} taking the left branch at split node i . Thus, if we have $y_{1,i} = 1$ for any first hidden layer node $i \in \mathcal{I}_{2,r}$, we know that the point would not have been sorted to leaf node r . In the case where $y_{1,i} = 1$ for any $i \in \mathcal{I}_{2,r}$, the maximum possible value of Eq. (12.42) is

$$1 - 2 - \frac{N_r - 1}{N_r} - \frac{1}{N_r + 1} < 0 \quad (12.45)$$

based on the weights we have defined in Eqs. (12.38) and (12.39), which again correctly has Eq. (12.44) holding true.

Lastly, if and only if $y_{1,i} = 1$ for all $i \in \mathcal{I}_{1,r}$ and $y_{1,i} = 0$ for every first hidden layer node $i \in \mathcal{I}_{2,r}$, we calculate that

$$\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} = \frac{1}{N_r} - \frac{1}{N_r + 1}, \quad (12.46)$$

meaning that

$$\mathbb{1}\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\} = 1. \quad (12.47)$$

This proves that node r in the second hidden layer of \mathcal{N}_7 activates if and only if a point is sorted to the corresponding leaf node in \mathcal{T}_7 for the case where $r \in C^*$.

If $r \notin C^*$, note that a point is sorted to such a leaf node if and only if $\mathbf{W}_i^T \mathbf{x} + b_i < 0$ for all $i \in \mathcal{I}_{2,r}$. If $\mathbf{W}_i^T \mathbf{x} + b_i \geq 0$ for any $i \in \mathcal{I}_{2,r}$, the the maximum possible value of Eq. (12.42) is

$$-1 + 0.1 < 0 \quad (12.48)$$

which correctly has

$$\mathbb{1}\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\} = 0. \quad (12.49)$$

If and only if $y_{1,i} = 0$ for every one of these nodes, we have

$$\mathbb{1}\{\mathbf{W}_{2,r}^T \mathbf{y}_1 + b_{2,r} \geq 0\} = \mathbb{1}\{0.1 \geq 0\} = 1. \quad (12.50)$$

This proves that node r in the second hidden layer of \mathcal{N}_7 activates if and only if a point is sorted to the corresponding leaf node in \mathcal{T}_7 for the case where $r \notin C^*$. Thus, by construction, we have that an arbitrary node r in \mathcal{N}_7 activates given input \mathbf{x} if and only if that input would be sorted to the corresponding leaf node in \mathcal{T}_7 .

Lastly, we need to define the output weights and biases $\mathbf{W}_{O,i}$ and $b_{O,i}$. In \mathcal{N}_7 we will model \mathcal{T}_7 's predicted class $k_{predicted}$ for input \mathbf{x} as output vector $\mathbf{y}_O(\mathbf{x})$, where $(\mathbf{y}_O(\mathbf{x}))_k = 1$ for $k = k_{predicted}$ and zero otherwise.

Then define the set of second hidden layer nodes in \mathcal{N}_7 where the corresponding leaf node r in \mathcal{T}_7 was assigned output value k , $k \in [q]$, as

$$\mathcal{R}_k = \{r \in [N_2] \mid \text{Leaf } r \text{ was assigned output } k\}, k \in [q]. \quad (12.51)$$

We then define

$$W_{O,r,k} = \begin{cases} 1, & \text{if } r \in \mathcal{R}_k, \\ 0, & \text{otherwise,} \end{cases} \quad (12.52)$$

$$b_{O,k} = -0.1. \quad (12.53)$$

We next show that the class \mathcal{N}_7 predicts for an input data point \mathbf{x} is the same as the class predicted by \mathcal{T}_7 . To see this, assume \mathbf{x} is input into \mathcal{T}_7 is sorted to leaf node r and assigned output value $k_{predicted}$. In \mathcal{N}_7 , by our definitions of \mathbf{W}_2 and \mathbf{b}_2 , we have $(\mathbf{y}_2(\mathbf{x}))_r = 1$ and is 0 elsewhere. By the above definitions in Eqs. (12.52) and (12.53),

$$(\mathbf{W}_O^T \mathbf{y}_2(\mathbf{x}) + \mathbf{b}_O)_{k_{predicted}} = 0.9 \geq 0 \quad (12.54)$$

while

$$(\mathbf{W}_O^T \mathbf{y}_2(\mathbf{x}) + \mathbf{b}_O)_k = -0.1 < 0 \text{ for } k \neq k_{predicted} \quad (12.55)$$

Thus, using the procedure described in Section 12.1, we find that the network and the tree predict the same classification value for a given data point \mathbf{x} , completing the proof. \square

12.6 Computational Results

In this section, we examine the performance of FNNs and OCT-Hs on seven datasets. We trained several different formulations of each model, and compared their out-of-sample accuracy on the seven datasets.

For each dataset, we trained and validated optimal trees of varying depths with regular splits and optimal trees with hyperplane splits of the appropriate depths on the datasets, using the procedure outlined in Algorithm 8.7. Once we had the best OCT and OCT-H based on the validation process, we calculated the models’ accuracy in classifying the data in the test set—this out-of-sample accuracy is included as the “Accuracy” value in the table. The tree depth is listed in the column labeled “Size Parameters.”

To train the neural networks, we used the TensorFlow library Tensorflow [249]. We used sigmoid functions here as the activation functions to make the networks easier to train, as they are a continuous approximation of the perceptron activation functions. We then trained and validated neural networks with sizes based on the tree depths we used based on the proof in Section 12.5—for example, a maximal tree with depth 2 has 3 split nodes and 4 leaf nodes, so for a neural network built with a size based on the tree we would have $N_1 = 3$ and $N_2 = 4$. The N_1 and N_2 values are listed in the column labeled “Size Parameters.” We validated the networks using the training parameters

$$\text{Step sizes} \in \{0.001, 0.01, 0.1, 1\}$$

$$\text{Regularization coefficients} \in \{1 \times 10^{-6}, 1 \times 10^{-5}, \dots, 0.1, 1\}$$

For each network size, we trained 50 neural networks with different random starts using grid search for the parameters. We then used the parameters with network with the best performance on a validation set to obtain the models’ accuracy in classifying the data in the test set. This out-of-sample accuracy is included as the “Accuracy” value in the table.

The seven data sets we use are an anonymized version of the Framingham heart study data set developed using the Teaching Dataset provided by the National Heart, Lung, and Blood Institute, and the Bank Marketing, Image Segmentation, Letter Recognition, Magic Gamma Telescope, Skin Segmentation, and Thyroid Disease ANN data sets from the UCI Machine Learning Repository [171]. In Table 12.3, we describe some details of the datasets we used.

Table 12.4 reports the results of our experiments

The results from Table 12.4 led to the following conclusions:

1. In six out of the seven datasets (the exception being Letter Recognition), the FNN and the OCT-H have very similar accuracy. Moreover, in these datasets OCT and OCT-H have also very similar accuracy.

Table 12.3: The datasets used and their parameters.

Dataset	# Parameters	# Class Values	# Data Points
Bank Marketing	17	2	45,211
Framingham Heart Study	15	2	3,658
Image Segmentation	18	7	210
Letter Recognition	16	26	20,000
Magic Gamma Telescope	10	2	19,020
Skin Segmentation	3	2	245,057
Thyroid Disease ANN	21	3	3,772

- word missing
- 2. The accuracy of the FNN was relatively insensitive to the size of the and similarly the accuracy of the OCT-H was relatively insensitive to the depth of the OCT-H.
 - 3. In Letter Recognition, OCT-H has a performance edge both with respect to FNN and OCT.

We have shown that OCT-Hs and FNNs are equivalent in terms of modeling power. However, the proofs require deep trees. These empirical results provide preliminary evidence that OCT-Hs (and OCTs) have comparable performance to neural networks even with small depth. This indicates that there is indeed practical merit to use OCT-Hs in practice rather than neural networks. The fact that OCTs gives similar results to FNNs is particularly noteworthy as OCTs are particularly interpretable.

12.7 Concluding Remarks

In this chapter, we showed that optimal decision trees are at least as powerful as neural networks in terms of modeling power and in the case of classification problems OCT-Hs and NNs have the same modeling power. While our constructions require deep trees that may be impractical to compute, we have also found that in seven datasets OCT-Hs and FNNs that the modeling power is indeed very similar even if the trees have small depth. While more empirical research is needed, we feel these findings are promising as OCT-Hs and especially OCTs are more interpretable than FNNs as they bring us closer to a significant objective of machine learning to build interpretable models with state of the art performance.

12.8 Notes and Sources

This chapter is based on [24]. We can trace the beginnings of neural networks to 1943. That year, [189] proposed to model neurons with simple electronic circuits, mirroring the fact that neurons, like circuits, either activate or not. Following this paper, there were more developments in the

Table 12.4: Accuracy of FNNs, OCT-Hs and OCTs.

Dataset	Model	Parameters	Test Results
Bank	FNN	$N_1 = 7, N_2 = 8, q = 2$	89.6%
	FNN	$N_1 = 15, N_2 = 16, q = 2$	89.4%
	FNN	$N_1 = 63, N_2 = 64, q = 2$	89.6%
	FNN	$N_1 = 255, N_2 = 256, q = 2$	89.6%
	OCT	max depth = 4, chosen depth 4	89.3%
	OCT	max depth = 6, chosen depth 6	89.6%
	OCT	max depth = 8, chosen depth 6	89.6%
	OCT-H	max depth = 4, chosen depth 3	89.6%
	OCT-H	max depth = 6, chosen depth 3	89.6%
	OCT-H	max depth = 8, chosen depth 3	89.6%
Framingham	FNN	$N_1 = 3, N_2 = 4, q = 2$	82.1%
	FNN	$N_1 = 15, N_2 = 16, q = 2$	82.1%
	FNN	$N_1 = 63, N_2 = 64, q = 2$	82.1%
	FNN	$N_1 = 255, N_2 = 256, q = 2$	81.7%
	OCT	max depth = 6, chosen depth 6	83.1%
	OCT	max depth = 8, chosen depth 8	82.4%
	OCT-H	max depth = 4, chosen depth 2	83.3%
	OCT-H	max depth = 6, chosen depth 2	83.3%
	OCT-H	max depth = 8, chosen depth 2	83.3%
Image Segmentation	FNN	$N_1 = 15, N_2 = 16, q = 7$	88.4%
	FNN	$N_1 = 31, N_2 = 32, q = 7$	83.7%
	FNN	$N_1 = 63, N_2 = 64, q = 7$	83.7%
	FNN	$N_1 = 255, N_2 = 256, q = 7$	83.7%
	OCT	max depth = 4, chosen depth 4	88.4%
	OCT	max depth = 6, chosen depth 6	88.4%
	OCT	max depth = 8, chosen depth 6	88.4%
	OCT-H	max depth = 4, chosen depth 4	86.0%
	OCT-H	max depth = 6, chosen depth 5	86.0%
	OCT-H	max depth = 8, chosen depth 5	86.0%
Letter Recognition	FNN	$N_1 = 15, N_2 = 16, q = 26$	58.6%
	FNN	$N_1 = 63, N_2 = 64, q = 26$	66.8%
	FNN	$N_1 = 255, N_2 = 256, q = 26$	67.0%
	OCT	max depth = 4, chosen depth 4	37.9%
	OCT	max depth = 6, chosen depth 6	59.1%
	OCT	max depth = 8, chosen depth 8	68.2%
	OCT-H	max depth = 4, chosen depth 4	44.6%
	OCT-H	max depth = 6, chosen depth 6	72.0%
	OCT-H	max depth = 8, chosen depth 8	80.3%

Table 12.4 continued: Accuracy of FNNs, OCT-Hs and OCTs.

Dataset	Model	Parameters	Test Results
MGT	FNN	$N_1 = 15, N_2 = 16, q = 2$	87.5%
	FNN	$N_1 = 31, N_2 = 32, q = 2$	88.4%
	FNN	$N_1 = 63, N_2 = 64, q = 2$	88.1%
	FNN	$N_1 = 255, N_2 = 256, q = 2$	88.3%
	OCT	max depth = 4, chosen depth 4	84.1%
	OCT	max depth = 6, chosen depth 6	85.3%
	OCT	max depth = 8, chosen depth 8	85.7%
	OCT-H	max depth = 4, chosen depth 4	86.7%
	OCT-H	max depth = 6, chosen depth 5	88.6%
	OCT-H	max depth = 8, chosen depth 5	87.0%
Skin Segmentation	FNN	$N_1 = 15, N_2 = 16, q = 2$	99.9%
	FNN	$N_1 = 63, N_2 = 64, q = 2$	99.9%
	FNN	$N_1 = 127, N_2 = 128, q = 2$	99.9%
	FNN	$N_1 = 255, N_2 = 256, q = 2$	99.9%
	OCT	max depth = 4, chosen depth 4	98.9%
	OCT	max depth = 6, chosen depth 6	99.8%
	OCT	max depth = 8, chosen depth 8	99.9%
	OCT-H	max depth = 4, chosen depth 4	99.9%
	OCT-H	max depth = 6, chosen depth 6	99.9%
	OCT-H	max depth = 8, chosen depth 7	99.9%
Thyroid	FNN	$N_1 = 7, N_2 = 8, q = 3$	97.7%
	FNN	$N_1 = 15, N_2 = 16, q = 3$	98.1%
	FNN	$N_1 = 63, N_2 = 64, q = 3$	97.4%
	FNN	$N_1 = 255, N_2 = 256, q = 3$	98.0%
	OCT	max depth = 4, chosen depth 4	99.7%
	OCT	max depth = 6, chosen depth 4	99.7%
	OCT	max depth = 8, chosen depth 4	99.7%
	OCT-H	max depth = 4, chosen depth 3	99.9%
	OCT-H	max depth = 6, chosen depth 3	99.9%
	OCT-H	max depth = 8, chosen depth 3	99.9%

field, such as the creation of a system that could learn how to classify input data known as the Perceptron [230], the development of backpropagation, a technique to train neural networks [268, 234], the proof that multilayer feedforward neural networks were universal approximators [165, 142], and many more. Increased computational power, advances in optimization (stochastic gradient methods), and the massive availability of datasets have also lead to the development of a methodology known as deep learning, which involves training large neural networks with many hidden layers. For a survey in developments in neural networks and deep learning, see [226, 167, 239].

Part III

Prescriptive Analytics

Chapter 13

From Predictive to Prescriptive Analytics

—



Chapter 14

Optimal Prescription Trees

In God we trust. All others must bring data.

– W. Edwards Deming

Contents

- 14.1. Optimal Prescriptive Trees
- 14.2. Experiments with Synthetic Datasets
- 14.3. Experiments with Real-World Datasets
- 14.4. Concluding Remarks
- 14.5. Notes and Sources

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

$$\sum_{i=1}^n \left(y_i \mathbb{I}[\tau(\mathbf{x}_i) = z_i] + \sum_{t \neq z_i} \hat{y}_i(t) \mathbb{I}[\tau(\mathbf{x}_i) = t] \right), \quad (14.1)$$

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 (14.1) 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

$$\left[\sum_{i=1}^n (y_i - \hat{y}_i(z_i))^2 \right], \quad (14.2)$$

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 (14.1) and (14.2):

$$\mu \left[\sum_{i=1}^n \left(y_i \mathbb{I}[\tau(\mathbf{x}_i) = z_i] + \sum_{t \neq z_i} \hat{y}_i(t) \mathbb{I}[\tau(\mathbf{x}_i) = t] \right) \right] + (1-\mu) \left[\sum_{i=1}^n (y_i - \hat{y}_i(z_i))^2 \right], \quad (14.3)$$

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 [96]), and more recently by researchers in statistics [218] and operations research [40]. The online version of this problem, called the contextual bandit problem, has been studied by several authors [169, 113, 11] in the multi-armed bandit literature [112]. 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. [40] 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 [28] 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 [29] 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 [229]. 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. [282] 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 [269] or as primarily covariate balancing [149], 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 [153] proposes personalization trees (and forests), which adapt regular classification trees [54] 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 8–11 to design near optimal trees, which improves performance significantly. Athey and Imbens [4] and Wager and Athey [265] 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. [217], 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 [175]. In an example of breast cancer, Gort et al. [119] 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 [100] and mental illness [150], 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 8–11 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. [10] 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 [214]. On the other hand, two arm trials can force the investigator to make potentially incorrect series of decisions on treatment, dose or assessment duration [214]. 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 [62]. 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.

14.1 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 (14.3) 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 $\hat{y}_i(t)$:

$$\hat{y}_i(t) = \frac{1}{|\{j : \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t\}|} \sum_{j: \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t} y_j, \quad (14.4)$$

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

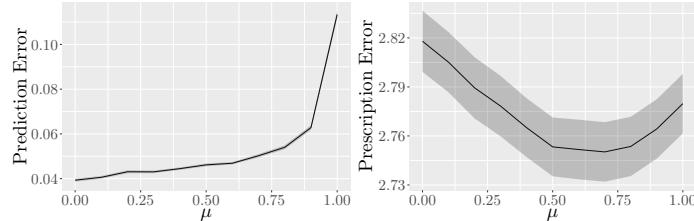
Substituting this back into Problem (14.3), we want to find a prescriptive tree τ that solves the following problem:

$$\begin{aligned} \min_{\tau(\cdot)} \quad & \mu \left[\sum_{i=1}^n \left(y_i \mathbb{I}[\tau(\mathbf{x}_i) = z_i] + \sum_{t \neq z_i} \frac{\sum_{j: \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t} y_j}{|\{j : \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = t\}|} \mathbb{I}[\tau(\mathbf{x}_i) = t] \right) \right] \\ & + (1 - \mu) \left[\sum_{i=1}^n \left(y_i - \frac{1}{|\{j : \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = z_i\}|} \sum_{j: \mathbf{x}_j \in \mathcal{X}_{l(i)}, z_j = z_i} y_j \right)^2 \right]. \end{aligned} \quad (14.5)$$

We note that when $\mu = 1$, we obtain the same objective function as Kallus [153], 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 (14.5) to global optimality using a MIO formulation based on an earlier version of Optimal Trees [26]. 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 this book, is practical and scales to large datasets while solving in tractable times. When $\mu = 0$, we obtain the objective function in Bertsimas and Dunn [26] 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.

Figure 14.1: Test prediction and personalization error as a function of μ



To illustrate this observation, Figure 14.1 shows the average prediction and prescription errors as a function of μ for one of the synthetic experiments we conduct in Section 14.2. 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 (14.5) and find Optimal Prescriptive Trees. The core of the algorithm remains as described in Sections 8.3 and 9.2 for parallel and hyperplane splits, respectively, and we set Problem (14.5) as the loss function to be optimized in 8.25. 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 (14.4). 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 11, 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 11 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 β_ℓ^t :

$$\min_{\beta_\ell^t} \frac{1}{2 |\{i : \mathbf{x}_i \in \mathcal{X}_\ell, z_i = t\}|} \sum_{i: \mathbf{x}_i \in \mathcal{X}_\ell, z_i = t} \left(y_i - (\beta_\ell^t)^T \mathbf{x}_i \right)^2 + \lambda P_\alpha(\beta_\ell^t), \quad (14.6)$$

where

$$P_\alpha(\beta) = (1 - \alpha) \frac{1}{2} \|\beta\|_2^2 + \alpha \|\vec{\beta}\|_1. \quad (14.7)$$

We proceed to estimate the counterfactuals with the following equation:

$$\hat{y}_i(t) = (\beta_{\ell(i)}^t)^T \mathbf{x}_i, \quad (14.8)$$

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

$$\begin{aligned} \min_{\tau(\cdot), \beta} \quad & \mu \left[\sum_{i=1}^n \left(y_i \mathbb{I}[\tau(\mathbf{x}_i) = z_i] + \sum_{t \neq z_i} (\beta_{\ell(i)}^t)^T \mathbf{x}_i \mathbb{I}[\tau(\mathbf{x}_i) = t] \right) \right] \\ & + (1 - \mu) \left[\sum_{i=1}^n \left(y_i - (\beta_{\ell(i)}^t)^T \mathbf{x}_i \right)^2 + \lambda \sum_{t=1}^m \sum_{\ell} P_{\alpha}(\beta_{\ell}^t) \right], \end{aligned} \quad (14.9)$$

where the regression models β are found by solving the elastic net problems (14.6) 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 β coefficients in the same way as standard regression. We solve this problem using the approach from 11.2, 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 (14.7). 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.

14.2 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. [217]. We generate n data points $\mathbf{x}_i, i = 1, \dots, n$ 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 $x_{ij} \sim \text{Bernoulli}(0.5)$.

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

$$\begin{aligned} y_0(x) &= \text{baseline}(\mathbf{x}) - \frac{1}{2}\text{effect}(\mathbf{x}), \\ y_1(x) &= \text{baseline}(\mathbf{x}) + \frac{1}{2}\text{effect}(\mathbf{x}). \end{aligned}$$

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:

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

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

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

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

$$\text{baseline}(\mathbf{x}) = 5\mathbb{1}(x_1 > 1) - 5,$$

$$\text{effect}(\mathbf{x}) = \frac{1}{2}(x_1^2 + x_2 + x_3^2 + x_4 + x_5^2 + x_6 + x_7^2 + x_8 + x_9^2 - 11).$$

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:

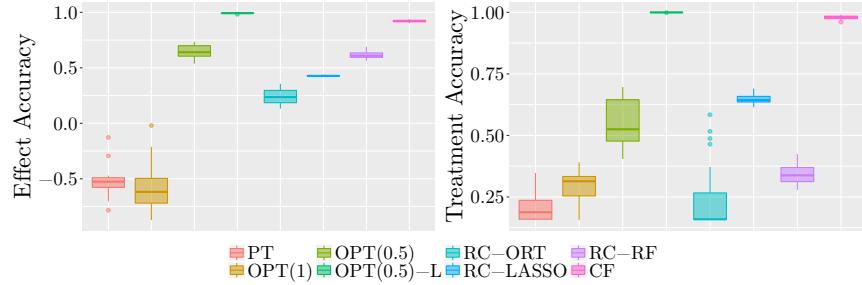
- *Treatment Accuracy*: the proportion of the test set where the prescriptions are correct;
- *Effect Accuracy*: the R^2 of the predicted effects, $y(1) - y(0)$, made by the model for each observation in the test set, compared against the true effect for each observation.

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.

Figure 14.2: Performance results for Experiment 1.

- **Causal Methods:** We include the method of causal forests [265] 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 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 14.2 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 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

Figure 14.3: Tree constructed by OPT(0.5)-L for an instance of Experiment 1.

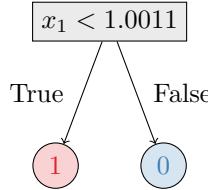
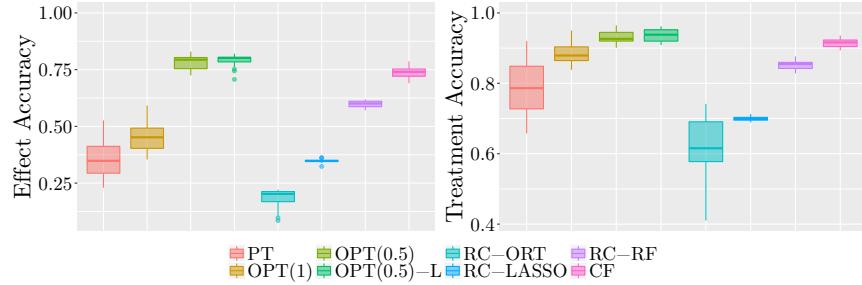


Figure 14.4: Performance results for Experiment 2.



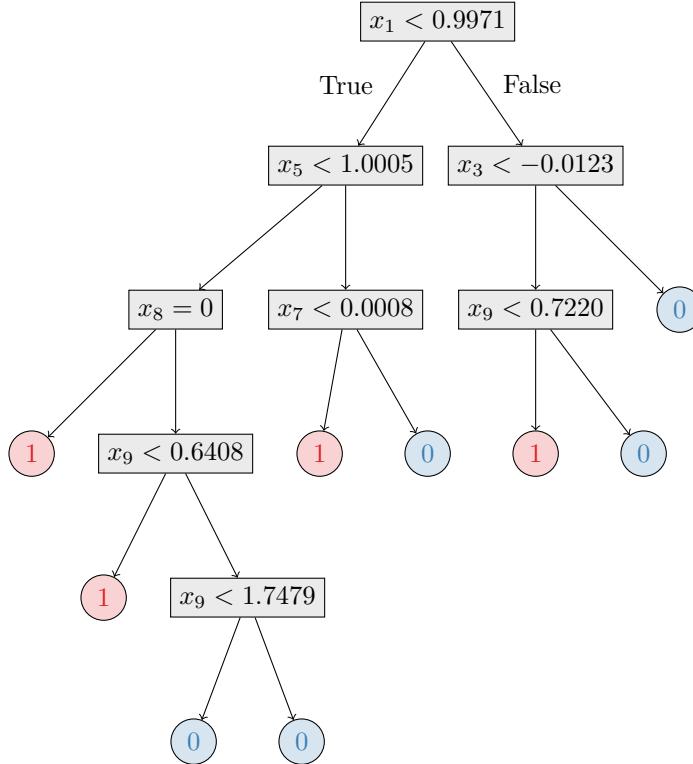
prescriptive and predictive components with the prescriptive factor μ in the objective function.

Figure 14.3 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 14.4. 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

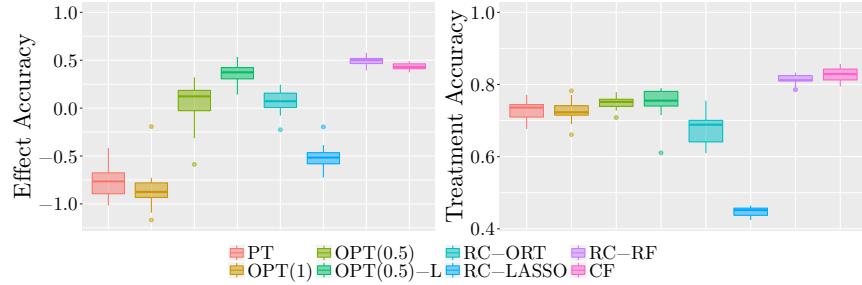
Figure 14.5: Tree constructed by OPT(0.5)-L for an instance of Experiment 2.



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 14.5. 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

Figure 14.6: Performance results for Experiment 3.

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 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 14.6 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 14.3 and 14.5, 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.

14.3 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 [74]. 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 [151].

The publicly available dataset we use was collected and curated by staff at the Pharmacogenetics and Pharmacogenomics Knowledge Base (PharmKG) 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

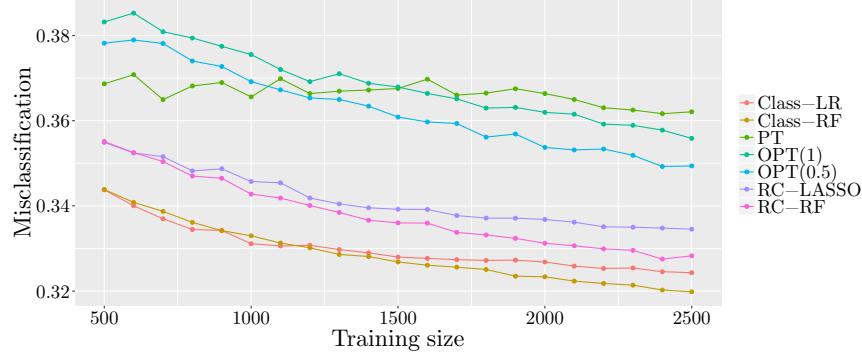
Since it is unlikely that a real world dataset will consist of these optimal prescriptions, we reassign some patients in the training set to other treatments so that their assignment is no longer optimal. To achieve this, we follow the setup of Kallus [153], and assume that the doctor prescribes warfarin dosage according to the following probabilistic assignment model:

$$\mathbb{P}(Z = t | \mathbf{X} = \mathbf{x}) = \frac{1}{S} \exp\left(\frac{(t - 1)(\text{BMI} - \mu)}{\sigma}\right), \quad (14.10)$$

where μ, σ are the population mean and standard deviation of patients' BMI respectively, and the normalizing factor

$$S = \sum_{t=1}^3 \exp\left(\frac{(t - 1)(\text{BMI} - \mu)}{\sigma}\right).$$

Figure 14.7: Misclassification rate for warfarin dosing prescriptions as a function of training set size.



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) , $i = 1, \dots, n$. 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 $h = 500, 600, \dots, 2500$, 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 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 14.2, 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 [58], denoted Class-RF, and logistic regression, denoted Class-LR.

In Figure 14.7, 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 [153], 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. [40], where the authors propose a k -nearest neighbors (k NN) 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 k NN approach.

We follow the same experimental design as in Bertsimas et al. [40]. 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 k NN 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.

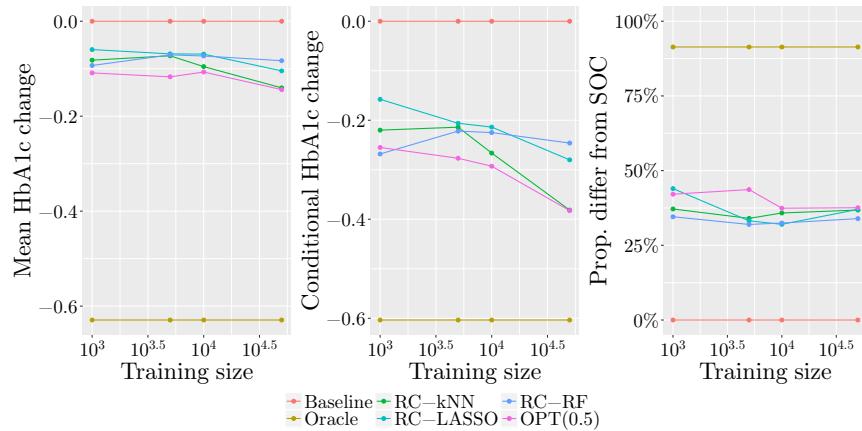
In addition to methods defined in Section 14.2, we compare the following approaches:

- **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- k NN, to match the approaches used in [40]

The results of the experiments are shown in Figure 14.8. We see that our results for the regress-and-compare methods mirror those of [40]; RC- k NN 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 k NN. 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- k NN, 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- k NN 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- k NN, and

Figure 14.8: 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.



significantly stronger at smaller training set sizes. The other critical advantage is the increased interpretability of OPT compared to RC- k NN, which is itself already more interpretable than other regress-and-compare approaches. To interpret the RC- k NN 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 [166], 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 program¹. 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–

¹ Available at <http://users.nber.org/~rdehejia/nswdata2.html>

Table 14.1: 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- k NN	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

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.

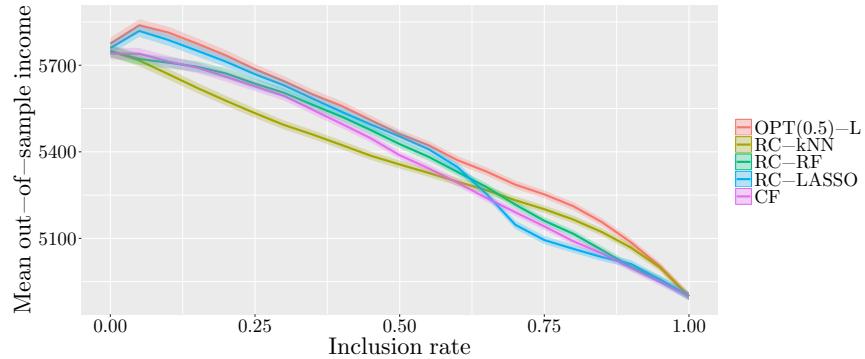
We split the full dataset into 70/30 training/testing samples, and averaged the results over 1000 such splits to plot the out of sample average personalized income. Since the counterfactuals are not known for this example we again employ the nearest-neighbor matching algorithm from [40] to impute the 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 14.1, 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 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 14.9, 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

Figure 14.9: Out-of-sample average personalized income as a function of inclusion rate.



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 [137]. Following Hill [137], 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 [137]). The outcomes are simulated in such a way that the average treatment effect on the control subjects is positive (setting B in Hill [137] 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

Table 14.2: 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

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 14.2, 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 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.

14.4 Concluding Remarks

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.

14.5 Notes and Sources

The material in this chapter is from [22].

Part IV

Unsupervised Methods

Chapter 15

Optimal Missing Data Imputations

Torture the data, and it will confess to anything.

– Ronald Coase, British economist and author

Contents

- 15.1. Methods for Optimal Imputation
- 15.2. K -NN Based Imputation
- 15.3. Mixed SVM Based Imputation
- 15.4. Tree Based Imputation
- 15.5. Model Selection
- 15.6. Real-World Data Experiments
- 15.7. Concluding Remarks
- 15.8. Notes and Sources

The missing data problem is arguably the most common issue encountered by machine learning practitioners when analyzing real-world data. In many applications ranging from gene expression in computational biology to survey responses in social sciences, missing data is present to various degrees. As many statistical models and machine learning algorithms rely on complete data sets, it is key to handle the missing data appropriately.

Given data $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with missing entries $x_{id}, (i, d) \in \mathcal{M}$, the objective is to impute the values of the missing data that resemble the underlying complete data as closely as possible. This way, when one conducts statistical inference or pattern recognition using machine learning methods on the imputed data, the results should be similar to those obtained if full data were given. The problem of missing data imputation has received extensive attention in the literature. We review some of the state-of-the-art methods for imputation in Section 15.8.

In this chapter, we pose the missing data problem under a general optimization framework. The framework produces an optimization problem with a predictive model-based cost function that explicitly handles both continuous and categorical variables. We present three cost functions derived from K -nearest neighbors, support vector machines, and optimal decision tree models. This optimization perspective provides insight into the classical missing data problem and leads to new algorithms for more accurate data imputation.

For each imputation model, we derive first-order methods to find high-quality solutions to the missing data problem following a general imputation algorithm `opt.impute` presented in this chapter. These methods scale to data sets with n in the 100,000s and p in the 1,000s on a standard desktop computer and converge within a few iterations. In addition, the first-order methods are robust and reliable for arbitrary missing patterns and mixed data types.

We evaluate the methods in computational experiments using 84 real-world data sets taken from the UCI Machine Learning Repository. Benchmarked against existing imputation methods including mean impute, K -nearest neighbors, iterative knn, Bayesian PCA, and predictive-mean matching, `opt.impute` produces the best overall imputation in more than 75.8% of all data sets, and results in an average reduction in mean absolute error of 8.3% against the best cross-validated benchmark method. We demonstrate that the improved data imputations generated by `opt.impute` give rise to improved performance on 10 downstream classification and regression tasks. With 50% of missing data, classification models trained on data imputed via `opt.impute` have an average testing accuracy of 86.1% compared to 84.4% for the best cross-validated benchmark method. In addition, regression models trained on data imputed via `opt.impute` have an average out-of-sample R^2 value of 0.339 compared to 0.315 for the best cross-validated benchmark method. Finally, downstream models trained

on multiple imputations produced by `opt.impute` significantly outperform multiple imputations produced by `mice` in 3/5 missing data scenarios for classification and 5/5 scenarios for regression.

15.1 Methods for Optimal Imputation

In this section, we pose the missing data problem as an optimization problem in which we optimize the missing values in all data points and dimensions simultaneously. We introduce a general imputation framework on mixed data (continuous and categorical) based upon first-order methods applied to this problem. Within this framework, we use K -nearest neighbors, SVM, and decision tree based imputation as examples to define three specific optimization problems. For each problem, we present two first-order methods used to find high-quality solutions: block coordinate descent (BCD) and coordinate descent (CD).

Let $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ be the data set given with p variables. Without loss of generality, we assume each data vector \mathbf{x}_i contains continuous variables indexed by $d \in \{1, 2, \dots, p_0\}$ and categorical variables indexed by $d \in \{p_0 + 1, \dots, p_0 + p_1\}$ with $p_0 + p_1 = p$. As a pre-processing step, we transform all continuous variables to have unit standard deviation. We leave all categorical variables unchanged, and assume the d th categorical variable $d \in \{p_0 + 1, \dots, p_0 + p_1\}$ takes values among k_d classes. Note that if all data is continuous $p_0 = 0$, while if all data is categorical $p_1 = 0$. The missing and known values are specified by the following sets:

$$\begin{aligned}\mathcal{M}_0 &= \{(i, d) : \text{entry } x_{id} \text{ is missing}, 1 \leq d \leq p_0\}, \\ \mathcal{N}_0 &= \{(i, d) : \text{entry } x_{id} \text{ is known}, 1 \leq d \leq p_0\}, \\ \mathcal{M}_1 &= \{(i, d) : \text{entry } x_{id} \text{ is missing}, p_0 + 1 \leq d \leq p_0 + p_1\}, \\ \mathcal{N}_1 &= \{(i, d) : \text{entry } x_{id} \text{ is known}, p_0 + 1 \leq d \leq p_0 + p_1\}.\end{aligned}$$

We also refer to the full missing pattern as $\mathcal{M} := \mathcal{M}_0 \cup \mathcal{M}_1$. Let $\mathbf{W} \in \mathbb{R}^{n \times p_0}$ be the matrix of imputed continuous values, where w_{id} is the imputed value for entry x_{id} , $d \in \{1, \dots, p_0\}$. Similarly, let $\mathbf{V} \in \{1, \dots, k_1\} \times \dots \times \{1, \dots, k_{p_1}\}$ be the matrix of imputed categorical values, where v_{id} is the imputed value for entry x_{id} , $d \in \{p_0 + 1, \dots, p_0 + p_1\}$. We refer to the full imputation for observation \mathbf{x}_i as $(\mathbf{w}_i, \mathbf{v}_i)$ in the following sections.

General Problem Formulation

As the task is to impute the missing values, for each model the key decision variables are the imputed values $\{w_{id} : (i, d) \in \mathcal{M}_0\}$ and $\{v_{id} : (i, d) \in \mathcal{M}_1\}$. We also introduce auxiliary decision variables as well; denote these as \mathbf{U} . For instance, in a K -NN based approach, indicator variables $z_{ij}, 1 \leq i, j \leq n$ are introduced to identify the neighbor assignment for each pair of

points \mathbf{x}_i , \mathbf{x}_j . For a given set of imputed values and a given model, there is a cost function $c(\cdot)$ associated with it. Our goal is to solve the following optimization problem:

$$\begin{aligned} \min \quad & c(\mathbf{U}, \mathbf{W}, \mathbf{V}; \mathbf{X}) \\ \text{s.t.} \quad & w_{id} = x_{id} \quad (i, d) \in \mathcal{N}_0, \\ & v_{id} = x_{id} \quad (i, d) \in \mathcal{N}_1, \\ & (\mathbf{U}, \mathbf{W}, \mathbf{V}) \in \mathcal{U}, \end{aligned} \tag{15.1}$$

where \mathcal{U} is the set of all feasible combinations $(\mathbf{U}, \mathbf{W}, \mathbf{V})$ of auxiliary vectors and imputations. For example, in a K -NN based approach, this includes the constraints that each point has exactly K neighbors and the assignment variables are binary. We list the auxiliary variables and cost functions corresponding to each of the imputation models K -NN, SVM, and trees in Table 15.1. Note that the cost function can be different for continuous and categorical variables. We can introduce a parameter that controls the relative contribution to the cost between the continuous and categorical variables, or scale continuous variables appropriately. For the remainder of the chapter the latter is assumed for simplicity of notation.

Table 15.1: Variables and cost functions for each imputation model. Variables for K -NN, SVM, and trees are defined in Sections 15.2, 15.3, and 15.4 respectively.

Model	\mathbf{U}	$c(\mathbf{U}, \mathbf{W}, \mathbf{V}; \mathbf{X})$
K -NN	\mathbf{Z}	$\sum_{i \in \mathcal{I}} \sum_{j=1}^n z_{ij} \left[\sum_{d=1}^{p_0} (w_{id} - w_{jd})^2 + \sum_{d=p_0+1}^{p_0+p_1} \mathbb{1}\{v_{id} \neq v_{jd}\} \right]$
SVM	$[\boldsymbol{\beta}, \boldsymbol{\theta}, \boldsymbol{\gamma}, \boldsymbol{\gamma}^*, \boldsymbol{\xi}]$	$\frac{1}{2} (\ \boldsymbol{\beta}\ _{\mathcal{H}}^2 + \ \boldsymbol{\theta}\ _{\mathcal{H}}^2) + C \sum_{i=1}^n \left(\sum_{d=1}^{p_0} (\gamma_{id} + \gamma_{id}^*) + \sum_{d=p_0+1}^{p_0+p_1} \xi_{id} \right)$
Trees	\mathbf{T}	$\sum_{i=1}^n \sum_{j=1}^n \left[\sum_{d=1}^{p_0} t_{ij}^d (w_{id} - w_{jd})^2 + \sum_{d=p_0+1}^{p_0+p_1} t_{ij}^d \mathbb{1}\{v_{id} \neq v_{jd}\} \right]$

This problem is non-convex for K -NN, SVM, and tree models. To obtain a certifiable optimal solution, one can reformulate the problem with integer variables and solve it using a mixed integer solver. We ran computational experiments and found that solving such mixed integer problems requires a long time to reach a certifiably optimal solution. As

a result, we present a general imputation algorithm `opt.impute` which approximates the solution to Problem (15.1) very fast using first-order methods.

First-Order Method for the General Problem

To obtain high-quality solutions to Problem (15.1), we can use first-order methods with random warm starts. In particular, we will focus on block coordinate descent (BCD) and coordinate descent (CD) [18]. Algorithm 15.1, which we refer to as `opt.impute`, implements BCD or CD for Problem (15.1). The variables \mathbf{U} , \mathbf{W} , \mathbf{V} , and \mathbf{X} as well as the cost function $c(\cdot)$ are summarized in Table 15.1 for K-NN, SVM, and trees. The detailed solution methods for Problems (15.2), (15.3), (15.4), and (15.5) for K-NN, SVM, and tree imputation models are described in Sections 15.2–15.4, respectively.

By construction, the objective function value strictly decreases by at least δ_0 until termination. It follows that the number of steps needed for the algorithm to terminate is $\lceil \frac{1}{\delta_0} c(\mathbf{U}^0, \mathbf{W}^0, \mathbf{V}^0; \mathbf{X}) \rceil$, where $\mathbf{W}^0, \mathbf{V}^0$ are the initialization values, \mathbf{X} is data, and \mathbf{U}^0 is the argmin in Equation (15.2). However, the algorithm is not guaranteed to find a global minimum for Problem (15.1) [274].

In the next sections, we discuss three examples of the optimization framework in this section and derive the specific updates for $\mathbf{U}, \mathbf{W}, \mathbf{V}$ that we use. We next describe a cross-validation procedure to select the specific model and parameters for the imputation.

15.2 K-NN Based Imputation

We first define a distance metric between rows $(\mathbf{w}_i, \mathbf{v}_i)$ and $(\mathbf{w}_j, \mathbf{v}_j)$ as

$$d_{ij} := \sum_{d=1}^{p_0} (w_{id} - w_{jd})^2 + \sum_{d=p_0+1}^{p_0+p_1} \mathbb{1}\{v_{id} \neq v_{jd}\}. \quad (15.6)$$

Next, we introduce the binary variables:

$$z_{ij} = \begin{cases} 1, & \text{if } (\mathbf{w}_j, \mathbf{v}_j) \text{ is among the } K \text{-nearest neighbors of } (\mathbf{w}_i, \mathbf{v}_i) \\ & \text{with respect to distance metric (15.6),} \\ 0, & \text{otherwise.} \end{cases}$$

We further define the set of indices

$$\mathcal{I} := \{i : \mathbf{x}_i \text{ has at least one missing coordinate}\}.$$

Algorithm 15.1 opt.impute

Input: $\mathbf{X} \in \mathbb{R}^{n \times p_0} \times \{1, \dots, k_1\} \times \dots \times \{1, \dots, k_{p_1}\}$, a data matrix with some missing entries $\mathcal{M} = \{(i, d) : x_{id} \text{ is missing}\}$, $\delta_0 > 0$, and warm start $\mathbf{W}^0 \in \mathbb{R}^{n \times p_0}$, $\mathbf{V}^0 \in \{1, \dots, k_1\} \times \dots \times \{1, \dots, k_{p_1}\}$.

Output: \mathbf{X}^{imp} a full matrix with imputed values.

- 1: $\delta \leftarrow \infty$, $\mathbf{W}^{old} \leftarrow \mathbf{W}^0$, $\mathbf{V}^{old} \leftarrow \mathbf{V}^0$
- 2: **while** $\delta > \delta_0$ **do**
- 3: ① Update \mathbf{U}^* using model dependent information:

$$\begin{aligned} \mathbf{U}^* &\leftarrow \arg \min_{\mathbf{U}} c(\mathbf{U}, \mathbf{W}^{old}, \mathbf{V}^{old}; \mathbf{X}) \\ \text{s.t. } (\mathbf{U}, \mathbf{W}^{old}, \mathbf{V}^{old}) &\in \mathcal{U}. \end{aligned} \quad (15.2)$$

- 4: ② Update the imputation \mathbf{W}^* , \mathbf{V}^* , following either:
- 5: ②A block coordinate descent (BCD):

$$\begin{aligned} \mathbf{W}^*, \mathbf{V}^* &\leftarrow \arg \min_{\mathbf{W}, \mathbf{V}} c(\mathbf{U}^*, \mathbf{W}, \mathbf{V}; \mathbf{X}) \\ \text{s.t. } w_{id} &= x_{id} \quad (i, d) \in \mathcal{N}_0, \\ v_{id} &= x_{id} \quad (i, d) \in \mathcal{N}_1, \\ (\mathbf{U}^*, \mathbf{W}, \mathbf{V}) &\in \mathcal{U}. \end{aligned} \quad (15.3)$$

- 6: ②B coordinate descent (CD):

$$\begin{aligned} w_{jr}^* &\leftarrow \arg \min_{w_{jr}} c(\mathbf{U}^*, \mathbf{W}, \mathbf{V}; \mathbf{X}) \\ \text{s.t. } w_{id} &= x_{id} \quad (i, d) \in \mathcal{N}_0, \\ v_{id} &= x_{id} \quad (i, d) \in \mathcal{N}_1, \\ w_{id} &= w_{id}^* \quad (i, d) \in \mathcal{M}_0 \setminus (j, r), \\ v_{id} &= v_{id}^* \quad (i, d) \in \mathcal{M}_1, \\ (\mathbf{U}^*, \mathbf{W}, \mathbf{V}) &\in \mathcal{U}. \end{aligned} \quad (15.4)$$

$$\begin{aligned} v_{jr}^* &\leftarrow \arg \min_{v_{jr}} c(\mathbf{U}^*, \mathbf{W}, \mathbf{V}; \mathbf{X}) \\ \text{s.t. } w_{id} &= x_{id} \quad (i, d) \in \mathcal{N}_0, \\ v_{id} &= x_{id} \quad (i, d) \in \mathcal{N}_1, \\ w_{id} &= w_{id}^* \quad (i, d) \in \mathcal{M}_0, \\ v_{id} &= v_{id}^* \quad (i, d) \in \mathcal{M}_1 \setminus (j, r), \\ (\mathbf{U}^*, \mathbf{W}, \mathbf{V}) &\in \mathcal{U}. \end{aligned} \quad (15.5)$$

- 7: ③ $\delta \leftarrow c(\mathbf{U}^*, \mathbf{W}^*, \mathbf{V}^*; \mathbf{X}) - c(\mathbf{U}^{old}, \mathbf{W}^{old}, \mathbf{V}^{old}; \mathbf{X})$.
- 8: ④ $(\mathbf{U}^{old}, \mathbf{W}^{old}, \mathbf{V}^{old}) \leftarrow (\mathbf{U}^*, \mathbf{W}^*, \mathbf{V}^*)$.
- 9: $\mathbf{X}^{imp} \leftarrow [\mathbf{W}^*; \mathbf{V}^*]$

The optimization problem for the K -NN based imputation model is:

$$\begin{aligned}
 \min \quad & c(\mathbf{Z}, \mathbf{W}, \mathbf{V}; \mathbf{X}) := \sum_{i \in \mathcal{I}} \sum_{j=1}^n z_{ij} \left[\sum_{d=1}^{p_0} (w_{id} - w_{jd})^2 + \sum_{d=p_0+1}^{p_0+p_1} \mathbb{1}\{v_{id} \neq v_{jd}\} \right] \\
 \text{s.t.} \quad & w_{id} = x_{id}, \quad (i, d) \in \mathcal{N}_0, \\
 & v_{id} = x_{id}, \quad (i, d) \in \mathcal{N}_1, \\
 & z_{ii} = 0, \quad i \in \mathcal{I}, \\
 & \sum_{j=1}^n z_{ij} = K, \quad i \in \mathcal{I}, \\
 & \mathbf{Z} \in \{0, 1\}^{|\mathcal{I}| \times n}
 \end{aligned} \tag{15.7}$$

By optimality, it follows that $z_{ij} = 1$ if and only if $(\mathbf{w}_j, \mathbf{v}_j)$ is among the K -nearest neighbors of $(\mathbf{w}_i, \mathbf{v}_i)$. Therefore, solving Problem (15.7) produces the missing value imputation which minimizes the sum of distances from each point $(\mathbf{w}_i, \mathbf{v}_i), i \in \mathcal{I}$ to its K -nearest neighbors. Note that the relation $\mathbb{1}\{v_{id} \neq v_{jd}\}$ can be modeled with binary variables. Problem (15.7) is a nonconvex optimization problem with both continuous and binary variables. Correspondingly, it is difficult to solve to provable optimality, even if the data set contains continuous variables only.

Next, we describe the updates in Algorithm 15.1 for K -NN based imputation. We refer to this specific imputation method as `opt.knn`.

`opt.knn`

In Step ①, to update the auxiliary variables \mathbf{Z} , first fix all imputed values \mathbf{W}, \mathbf{V} . Problem (15.2) decomposes by $i \in \mathcal{I}$ into the assignment problems:

$$\begin{aligned}
 \min_{\mathbf{z}_i} \quad & \sum_{j=1}^n z_{ij} d_{ij} \\
 \text{s.t.} \quad & z_{ii} = 0, \\
 & \sum_{j=1}^n z_{ij} = K, \\
 & \mathbf{z}_i \in \{0, 1\}^n
 \end{aligned} \tag{15.8}$$

The optimal solution to Problem (15.8) can be found using a simple sorting procedure on the distances $\{d_{ij}\}_{j=1}^n$. For each $i \in \mathcal{I}$, we find the K -nearest neighbors of $(\mathbf{w}_i, \mathbf{v}_i)$ and set $z_{ij} = 1$ for these neighbors, $z_{ij} = 0$, otherwise.

Next, we fix \mathbf{Z} and update the imputed values \mathbf{W}, \mathbf{V} using either BCD or CD. In Step ②a), the BCD update, Problem (15.3) decomposes by dimension $d = 1, \dots, p$. For each continuous dimension $d = 1, \dots, p_0$, we

consider the following quadratic optimization problem:

$$\begin{aligned} \min_{\mathbf{w}^d} \quad & \sum_{i \in \mathcal{I}} \sum_{j=1}^n z_{ij} (w_{id} - w_{jd})^2 \\ \text{s.t.} \quad & w_{id} = x_{id} \quad (i, d) \in \mathcal{N}_0, \end{aligned}$$

where $\mathbf{w}^d \in \mathbb{R}^n$ are the imputed values in the d th dimension. Taking partial derivative of the objective function with respect to w_{id} for some missing entry $(i, d) \in \mathcal{M}_0$ and setting it to zero, we obtain after some simplifications:

$$\begin{aligned} (K + \sum_{j \in \mathcal{I}} z_{ji}) w_{id} - \sum_{(j,d) \in \mathcal{M}_0} (z_{ij} + z_{ji}) w_{jd} \\ - \sum_{(j,d) \in \mathcal{N}_0} (z_{ij} + \mathbf{1}\{j \in \mathcal{I}\} z_{ji}) x_{jd} = 0. \end{aligned} \quad (15.9)$$

For each continuous dimension d , we have a system of equations of the form (15.9) which we can solve to determine the optimal imputed values $w_{id}, (i, d) \in \mathcal{M}_0$. To simplify notation, suppose that the missing values for dimension d are $\tilde{\mathbf{w}} := (\tilde{w}_{1d}, \dots, \tilde{w}_{ad})$ and the known values are $\tilde{\mathbf{x}} := (\tilde{x}_{(a+1)d}, \dots, \tilde{x}_{nd})$. Then, the set of optimal imputed missing values $\tilde{\mathbf{w}}$ is the solution to the linear system $\mathbf{Q}\tilde{\mathbf{w}} = \mathbf{R}\tilde{\mathbf{x}}$, where

$$\mathbf{Q} = \begin{bmatrix} K + \sum_{j \in \mathcal{I}} z_{j1} - 2z_{11} & -z_{12} - z_{21} & \dots & -z_{1a} - z_{a1} \\ -z_{21} - z_{12} & K + \sum_{j \in \mathcal{I}} z_{j2} - 2z_{22} & \dots & -z_{2a} - z_{a2} \\ \vdots & & \ddots & \vdots \\ -z_{a1} - z_{1a} & -z_{a2} - z_{2a} & \dots & K + \sum_{j \in \mathcal{I}} z_{ja} - 2z_{aa} \end{bmatrix},$$

$$\mathbf{R} = \begin{bmatrix} z_{1(a+1)} + \mathbf{1}\{(a+1) \in \mathcal{I}\} z_{(a+1)1} & \dots & z_{1n} + \mathbf{1}\{n \in \mathcal{I}\} z_{n1} \\ \vdots & & \vdots \\ z_{a(a+1)} + \mathbf{1}\{(a+1) \in \mathcal{I}\} z_{(a+1)a} & \dots & z_{an} + \mathbf{1}\{n \in \mathcal{I}\} z_{na} \end{bmatrix}.$$

Note that when K is sufficiently large, the matrix \mathbf{Q} is positive semidefinite and therefore invertible. If \mathbf{Q} is singular, then we may add a small positive perturbation to the diagonal of \mathbf{Q} so that the matrix becomes positive semidefinite. Therefore, without loss of generality there is a closed-form solution $\tilde{\mathbf{w}} = \mathbf{Q}^{-1}\mathbf{R}\tilde{\mathbf{x}}$ to this system of equations for each continuous dimension d .

In order to update \mathbf{V} , we solve the following integer linear optimization problem for each categorical dimension $d = (p_0 + 1), \dots, p$:

$$\begin{aligned} \min_{\mathbf{v}^d} \quad & \sum_{i \in \mathcal{I}} \sum_{j=1}^n z_{ij} y_{ij} \\ \text{s.t.} \quad & v_{id} = x_{id} \quad (i, d) \in \mathcal{N}_1, \\ & v_{id} - v_{jd} \leq y_{ij} k_d \quad i \in [n], j \in [n], \\ & v_{jd} - v_{id} \leq y_{ij} k_d \quad i \in [n], j \in [n], \\ & y_{ij} \in \{0, 1\}^{n \times n}, \end{aligned}$$

where $\mathbf{v}^d \in \{1, \dots, k_d\}^n$ are the imputed values for the d th dimension. Here, the indicator variables y_{ij} take values equal to $\mathbb{1}\{v_{jd} \neq v_{id}\}$ in the optimal solution.

In Step 2b, following the CD method, we update the missing imputed values one at a time. Each $w_{id}, (i, d) \in \mathcal{M}_0$ is imputed as the minimizer of the problem:

$$\min_{w_{id}} \quad \sum_{j=1}^n z_{ij} (w_{id} - w_{jd})^2 + \sum_{j \in \mathcal{I}} z_{ji} (w_{jd} - w_{id})^2,$$

leading to

$$w_{id} = \frac{\sum_{j=1}^n z_{ij} w_{jd} + \sum_{j \in \mathcal{I}} z_{ji} w_{jd}}{K + \sum_{j \in \mathcal{I}} z_{ji}}. \quad (15.10)$$

We can interpret the missing value imputation (15.10) as a weighted average of the K nearest neighbors of \mathbf{x}_i , along with all points \mathbf{x}_j which include \mathbf{x}_i as a neighbor. Similarly, each categorical variable $v_{id}, (i, d) \in \mathcal{M}_1$ is imputed as the minimizer of the problem:

$$\min_{v_{id}} \quad \sum_{j=1}^n z_{ij} \mathbb{1}\{v_{id} \neq v_{jd}\} + \sum_{j \in \mathcal{I}} z_{ji} \mathbb{1}\{v_{jd} \neq v_{id}\}.$$

The solution is

$$v_{id} = \text{mode}(\{\{v_{jd} : z_{ij} = 1\}, \{v_{jd} : z_{ji} = 1\}\}).$$

Here, we set v_{id} to be the highest frequency category among the K nearest neighbors of \mathbf{x}_i , along with all points \mathbf{x}_j which include \mathbf{x}_i as a nearest neighbor. In practice, we use this update for $v_{id}, (i, d) \in \mathcal{M}_1$ in place of the update for \mathbf{V} in BCD because it is much faster computationally.

15.3 Mixed SVM Based Imputation

In this section, we consider a second model for imputation, based upon SVM regression for imputing continuous features and SVM classification for imputing categorical features. First, we define $\tilde{\mathbf{v}}_i \in \{-1, 1\}^{p_2}$ to be a dummy encoded representation of \mathbf{v}_i , where $p_2 = \sum_{d=p_0+1}^{p_0+p_1} k_d - p_1$. Let $\tilde{v}_{id}^{fixed}, (i, d) \in \mathcal{N}_2$ be the known dummy encoded values. For each continuous feature $d \in \{1, \dots, p_0\}$, let $(\beta_d, \beta_{d0}) \in \mathbb{R}^{p_0+p_2+1}$ be the coefficients for an SVM regression model regressing feature d on the other features with the dummy encoding. Let $(\theta_d, \theta_{d0}) \in \mathbb{R}^{p_0+p_2+1}$ be the coefficients for an SVM classification model predicting dummy feature d based upon the other features. Note that it is also possible to use a multi-class SVM model to predict each categorical feature directly using parameters of the form $\mathbf{M} \in \mathbb{R}^{k_d \times (p_0+p_2+1)}$ for each feature $d \in \{p_0 + 1, \dots, p_0 + p_1\}$. In this case, we would keep the dummy encoded decision variables as covariates to predict the other features and add constraints relating $v_{id}, (i, d) \in \mathcal{M}_1$ and $\tilde{v}_{id}, (i, d) \in \mathcal{M}_2$. For illustrative purposes and simplicity of notation, we present the formulation using binary SVM to predict each dummy variable d .

We consider the following optimization problem:

$$\begin{aligned}
 \min \quad & c([\boldsymbol{\beta}, \boldsymbol{\theta}], \mathbf{W}, \tilde{\mathbf{V}}; \mathbf{X}) := \frac{1}{2} (\|\boldsymbol{\theta}\|^2 + \|\boldsymbol{\beta}\|^2) \\
 & + C \left(\sum_{i=1}^n \sum_{d=1}^{p_0} (\gamma_{id} + \gamma_{id}^*) + \sum_{i=1}^n \sum_{d=p_0+1}^{p_0+p_1} \xi_{id} \right) \\
 \text{s.t.} \quad & x_{id} = w_{id}, \quad (i, d) \in \mathcal{N}_0, \\
 & \tilde{v}_{id} = \tilde{v}_{id}^{fixed}, \quad (i, d) \in \mathcal{N}_2, \\
 & \beta_{dd} = 0, \quad d \in [p_0], \\
 & \theta_{dd} = 0, \quad d \in [p_2], \\
 & \gamma_{id} \geq w_{id} - (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - \epsilon, \quad d \in [p_0], i \in [n], \\
 & \gamma_{id}^* \geq (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - w_{id} - \epsilon, \quad d \in [p_0], i \in [n], \\
 & \xi_{id} \geq 1 - \tilde{v}_{id} (\boldsymbol{\theta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \theta_{d0}), \quad d \in [p_2], i \in [n], \\
 & \gamma_{id} \geq 0, \quad d \in [p_0], i \in [n], \\
 & \gamma_{id}^* \geq 0, \quad d \in [p_0], i \in [n], \\
 & \xi_{id} \geq 0, \quad d \in [p_2], i \in [n], \\
 & \tilde{v}_{id} \in \{-1, 1\}, \quad d \in [p_2], i \in [n].
 \end{aligned} \tag{15.11}$$

This formulation is based on SVM with a linear kernel. We can extend Problem (15.11) to arbitrary kernels, including the multi-class cases, using

the modified objective function

$$\begin{aligned} c([\boldsymbol{\beta}, \boldsymbol{\theta}], \mathbf{W}, \mathbf{V}; \mathbf{X}) := & \frac{1}{2}(\|\boldsymbol{\beta}\|_{\mathcal{H}}^2 + \|\boldsymbol{\theta}\|_{\mathcal{H}}^2) \\ & + C \left(\sum_{i=1}^n \sum_{d=1}^{p_0} (\gamma_{id} + \gamma_{id}^*) + \sum_{i=1}^n \sum_{d=p_0+1}^{p_0+p_1} \xi_{id} \right), \end{aligned}$$

where $\|\cdot\|_{\mathcal{H}}$ is the norm in a given Reproducing Kernel Hilbert Space \mathcal{H} .

Another important aspect of Problem (15.11) is the compound objective function, which is the summation of objective functions derived from both SVM regression and SVM classification methods. Observe that if we fix a single imputed entry w_{id} or \tilde{v}_{id} , the contribution to the objective function scales linearly as $(\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0})$ if d is continuous or scales linearly as $(\boldsymbol{\theta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \theta_{d0})$ if d is categorical. This is desirable because we do not wish to weight continuous and categorical variables unequally in our imputation. Next, we describe the updates in Algorithm 15.1 for mixed SVM based imputation, which we refer to as `opt.svm`.

`opt.svm`

In Step ①, we fix the imputed values \mathbf{W}, \mathbf{V} and update the auxiliary variables $[\boldsymbol{\beta}, \boldsymbol{\beta}_0, \boldsymbol{\theta}, \boldsymbol{\theta}_0]$. Independent of the choice of kernel, Problem (15.2) decomposes by dimension p into p_0 SVM regression problems and p_2 SVM classification problems for the categorical variables. For each continuous feature $d \in \{1, \dots, p_0\}$, we update $\boldsymbol{\beta}_d, \beta_{d0}$ by solving

$$\begin{aligned} \min \quad & \frac{1}{2} \|\boldsymbol{\beta}\|^2 + C \sum_{i=1}^n (\gamma_{id} + \gamma_{id}^*) \\ \text{s.t.} \quad & \beta_{dd} = 0 \\ & \gamma_{id} \geq w_{id} - (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - \epsilon \quad i \in [n], \\ & \gamma_{id}^* \geq (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - w_{id} - \epsilon \quad i \in [n], \\ & \gamma_{id} \geq 0 \quad i \in [n], \\ & \gamma_{id}^* \geq 0 \quad i \in [n]. \end{aligned} \tag{15.12}$$

Similarly, for each dummy feature $d \in \{p_0 + 1, \dots, p_0 + p_2\}$, we update θ_d, θ_{d0} by solving

$$\begin{aligned} \min \quad & \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{i=1}^n \xi_{id} \\ \text{s.t.} \quad & \theta_{dd} = 0 \\ & \xi_{id} \geq 1 - \tilde{v}_{id}(\boldsymbol{\theta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \theta_{d0}) \quad i \in [n], \\ & \xi_{id} \geq 0 \quad i \in [n]. \end{aligned} \tag{15.13}$$

Taking the Lagrangian duals, both Problems (15.12) and (15.13) can be reformulated as quadratic optimization problems which can be solved efficiently.

Next, we fix the auxiliary variables $[\boldsymbol{\beta}, \boldsymbol{\beta}_0, \boldsymbol{\theta}, \boldsymbol{\theta}_0]$ and update the imputed values \mathbf{W}, \mathbf{V} using BCD or CD. In Step (2a), Problem (15.2) decomposes by observation i into n nonlinear integer optimization problems. For each i we solve

$$\begin{aligned} \min_{\mathbf{w}_i, \tilde{\mathbf{v}}_i} \quad & \sum_{d=1}^{p_0} (\gamma_{id} + \gamma_{id}^*) + \sum_{d=p_0+1}^{p_0+p_1} \xi_{id} \\ \text{s.t.} \quad & x_{id} = w_{id} \quad (i, d) \in \mathcal{N}_0, \\ & \gamma_{id} \geq w_{id} - (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - \epsilon \quad d \in [p_0], \\ & \gamma_{id}^* \geq (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - w_{id} - \epsilon \quad d \in [p_0], \\ & \xi_{id} \geq 1 - \tilde{v}_{id}(\boldsymbol{\theta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \theta_{d0}) \quad d \in [p_2], \\ & \gamma_{id} \geq 0 \quad d \in [p_0], \\ & \gamma_{id}^* \geq 0 \quad d \in [p_0], \\ & \xi_{id} \geq 0 \quad d \in [p_2], \end{aligned} \tag{15.14}$$

where $(\mathbf{w}_i, \tilde{\mathbf{v}}_i) \in \mathbb{R}^{p_0} \times \{-1, 1\}^{p_2}$ is the imputation for observation \mathbf{x}_i . Note that if all features are continuous, Problem (15.14) reduces to a linear optimization problem. Because we are using the dummy encoding in this formulation, it is possible to obtain an imputation in which multiple classes are selected for a single categorical entry. In this case, when `opt.svm` terminates, we select the imputation among the set of potential candidates which minimizes the objective function of Problem (15.14).

In Step (2b), we update the imputed values one at a time. To update

$w_{id}, (i, d) \in \mathcal{M}_0$, we solve the one-dimensional linear optimization problem:

$$\begin{aligned} \min_{w_{id}} \quad & \sum_{d=1}^{p_0} (\gamma_{id} + \gamma_{id}^*) + \sum_{d=p_0+1}^{p_0+p_1} \xi_{id} \\ \text{s.t.} \quad & \gamma_{id} \geq w_{id} - (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - \epsilon \quad d \in [p_0], \\ & \gamma_{id}^* \geq (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - w_{id} - \epsilon \quad d \in [p_0], \\ & \xi_{id} \geq 1 - \tilde{v}_{id}(\boldsymbol{\theta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \theta_{d0}) \quad d \in [p_2], \\ & \gamma_{id} \geq 0 \quad d \in [p_0], \\ & \gamma_{id}^* \geq 0 \quad d \in [p_0], \\ & \xi_{id} \geq 0 \quad d \in [p_2]. \end{aligned}$$

We update $\tilde{v}_{id}, (i, d) \notin \mathcal{N}_2$ by solving the binary optimization problem:

$$\begin{aligned} \min_{\tilde{v}_{id} \in \{-1, 1\}} \quad & \sum_{i=1}^n \sum_{d=1}^{p_0} (\max\{w_{id} - (\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - \epsilon, 0\} + \\ & \max\{(\boldsymbol{\beta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \beta_{d0}) - w_{id} - \epsilon, 0\}) + \\ & \sum_{i=1}^n \sum_{d=1}^{p_2} (1 - \tilde{v}_{id}(\boldsymbol{\theta}_d^T \begin{bmatrix} \mathbf{w}_i \\ \tilde{\mathbf{v}}_i \end{bmatrix} + \theta_{d0})). \end{aligned}$$

15.4 Tree Based Imputation

In this section, we consider an imputation model based on optimal classification and regression trees. For each dimension we train a decision tree to predict the missing values, using the other features as covariates. We train regression trees to predict each of the continuous variables and classification trees to predict each of the categorical variables. Given a regression tree for continuous dimension d , we will impute $x_{id}, (i, d) \in \mathcal{M}_0$ to be the mean in dimension d of all points in the same leaf node as \mathbf{x}_i . Similarly, given a classification tree for dimension d , we will impute $x_{id}, (i, d) \in \mathcal{M}_1$ to be the mode in dimension d of all points in the same leaf node as \mathbf{x}_i .

For general prediction tasks, we use optimal classification or regression trees to train the decision trees. For each dimension d , let $\mathbf{T}^d \in \{0, 1\}^{n \times n}$ denote the set of indicator variables

$$t_{ij}^d = \begin{cases} 1, & \text{if } (\mathbf{w}_i, \mathbf{v}_i), (\mathbf{w}_j, \mathbf{v}_j) \text{ are in the same leaf node} \\ & \text{of the decision tree for dimension } d, \\ 0, & \text{otherwise.} \end{cases}$$

Let $(\mathbf{T}^d, \mathbf{W}, \mathbf{V}) \in \mathcal{T}^d$ denote the set of optimal decision tree constraints for dimension d as described in [26]. We consider the following optimization problem:

$$\begin{aligned} \min \quad & c(\mathbf{T}, \mathbf{W}, \mathbf{V}; \mathbf{X}) := \\ & \sum_{i=1}^n \sum_{j=1}^n \left[\sum_{d=1}^{p_0} t_{ij}^d (w_{id} - w_{jd})^2 + \sum_{d=p_0+1}^{p_0+p_1} t_{ij}^d \mathbf{1}\{v_{id} \neq v_{jd}\} \right] \\ \text{s.t. } & w_{id} = x_{id}, \quad (i, d) \in \mathcal{N}_0, \\ & v_{id} = x_{id}, \quad (i, d) \in \mathcal{N}_1, \\ & (\mathbf{T}^d, \mathbf{W}, \mathbf{V}) \in \mathcal{T}^d, \quad d \in [p]. \end{aligned} \quad (15.15)$$

Next, we describe the updates in Algorithm 15.1 for decision tree based imputation, which we refer to as `opt.tree`.

`opt.tree`

In Step ①, we fix the imputed values \mathbf{W}, \mathbf{V} and update the decision tree variables \mathbf{T} . For each continuous feature, we fit a regression tree to predict \mathbf{w}^d based upon the other features. Similarly, for each categorical feature, we fit an optimal classification tree to predict \mathbf{v}^d based upon the other features.

Next, we fix \mathbf{T} and update the imputed values \mathbf{W}, \mathbf{V} using BCD or CD. In Step ②a), Problem (15.3) decomposes by dimension into p_0 quadratic optimization problems and p_1 integer optimization problems. For each continuous dimension $d = 1, \dots, p_0$, we solve:

$$\begin{aligned} \min_{\mathbf{w}^d} \quad & \sum_{i=1}^n \sum_{j=1}^n t_{ij}^d (w_{id} - w_{jd})^2 \\ \text{s.t. } & w_{id} = x_{id} \quad (i, d) \in \mathcal{N}_0, \end{aligned}$$

where $\mathbf{w}^d \in \mathbb{R}^n$ are the imputed values in the d th dimension. This is a quadratic optimization problem with an explicit optimum. For each $w_{id}, (i, d) \in \mathcal{M}_0$, an optimal solution is

$$w_{id} = \begin{cases} \frac{\sum_{(j,d) \in \mathcal{N}_0^d} t_{ij}^d x_{jd}}{\sum_{(j,d) \in \mathcal{N}_0^d} t_{ij}^d}, & \text{if } \sum_{(j,d) \in \mathcal{N}_0^d} t_{ij}^d \geq 1, \\ \frac{1}{|\mathcal{N}_0^d|} \sum_{(j,d) \in \mathcal{N}_0^d} x_{jd}, & \text{otherwise,} \end{cases}.$$

where $\mathcal{N}_0^d := \{(i, r) \in \mathcal{N}_0 : r = d\}$. This solution corresponds to setting each missing entry equal to the mean of all observed values in the same leaf node. If the number of non-missing values in the same leaf node as w_{id} is zero, i.e., $\sum_{(j,d) \in \mathcal{N}_0^d} t_{ij}^d = 0$, then we set all of the values in that leaf node to the mean impute solution.

For each categorical dimension $d = p_0 + 1, \dots, p_0 + p_1$, we solve the following integer optimization problem:

$$\begin{aligned} \min_{\mathbf{v}^d} \quad & \sum_{i=1}^n \sum_{j=1}^n t_{ij}^d \mathbb{1}\{v_{id} \neq v_{jd}\} \\ \text{s.t.} \quad & v_{id} = x_{id}, \quad (i, d) \in \mathcal{N}_1, \end{aligned}$$

where $\mathbf{v}^d \in \{1, \dots, k_d\}^n$ are the imputed values for the d th dimension. An optimal solution is

$$v_{id} = \begin{cases} \text{mode}(\{x_{jd} : t_{ij}^d = 1, (j, d) \in \mathcal{N}_1\}), & \text{if } |\{x_{jd} : t_{ij}^d = 1, (j, d) \in \mathcal{N}_1\}| \geq 1, \\ \text{mode}(\{x_{jd} : (j, d) \in \mathcal{N}_1\}), & \text{otherwise.} \end{cases}.$$

In Step 2b, we update the missing imputed values one at a time, which results in slightly different closed form solutions for $w_{id}, (i, d) \in \mathcal{M}_0$ and $v_{id}, (i, d) \in \mathcal{M}_1$. First, we update the continuous variables $w_{id}, (i, d) \in \mathcal{M}_0$ by solving:

$$\min_{w_{id}} \quad \sum_{j=1}^n t_{ij}^d (w_{id} - w_{jd})^2. \quad (15.16)$$

An optimal solution to Problem (15.16) is

$$w_{id} = \begin{cases} \frac{\sum_{j \neq i} t_{ij}^d w_{jd}}{\sum_{j \neq i} t_{ij}^d}, & \text{if } \sum_{j \neq i} t_{ij}^d \geq 1, \\ \frac{1}{|\mathcal{N}_0^d|} \sum_{(j, d) \in \mathcal{N}_0^d} x_{jd}, & \text{otherwise.} \end{cases}$$

Next, we update the categorical variables $v_{id}, (i, d) \in \mathcal{M}_1$ one at a time by solving:

$$\min_{v_{id}} \quad \sum_{j=1}^n t_{ij}^d \mathbb{1}\{v_{id} \neq v_{jd}\}. \quad (15.17)$$

An optimal solution to Problem (15.17) is

$$v_{id} = \begin{cases} \text{mode}(\{v_{jd} : t_{ij}^d = 1\}), & \text{if } |\{v_{jd} : t_{ij}^d = 1\}| \geq 1, \\ \text{mode}(\{x_{jd} : (j, d) \in \mathcal{N}_1\}), & \text{otherwise.} \end{cases}$$

Both of these updates coincide with the predicted values from the decision trees constructed.

15.5 Model Selection

Given a choice of hyperparameters, each of the three methods we presented generates a single set of imputed values. Given \mathbf{X} with existing missing

Table 15.2: Hyperparameters tuned via the model selection procedure outlined in Section 15.5. σ^2 is a parameter in the radial basis function kernel, $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{\sigma^2}\right)$. cp is a complexity parameter related to the depth of the decision tree.

Method	Hyperparameters
K -NN	K
SVM	C, σ^2
Trees	cp

data $\mathcal{M}_0, \mathcal{M}_1$, we generate an additional fixed percentage of data missing $\mathcal{M}_0^{valid}, \mathcal{M}_1^{valid}$, with the known values as the hold-out set, and perform each of the imputation methods. We evaluate the imputation quality on the hold-out validation set by measuring how closely the imputed values resemble the ground truth values. In particular, the mean absolute error (MAE) between true and imputed values for each imputation method is calculated. The validation MAE is defined to be

$$\frac{1}{|\mathcal{M}_0^{valid}|} \sum_{(i,d) \in \mathcal{M}_0^{valid}} |w_{id} - x_{id}| + \frac{1}{|\mathcal{M}_1^{valid}|} \sum_{(i,d) \in \mathcal{M}_1^{valid}} \mathbb{1}\{v_{id} \neq x_{id}\}.$$

Lower values indicate closer imputation, and perfect imputation corresponds to an MAE of zero. Another metric of imputation quality is root mean squared error (RMSE), which is given by

$$\sqrt{\frac{1}{|\mathcal{M}_0^{valid}|} \sum_{(i,d) \in \mathcal{M}_0^{valid}} (w_{id} - x_{id})^2 + \frac{1}{|\mathcal{M}_1^{valid}|} \sum_{(i,d) \in \mathcal{M}_1^{valid}} \mathbb{1}\{v_{id} \neq x_{id}\}}.$$

For each imputation method, the combination of hyperparameters that achieves the lowest MAE in validation (or RMSE) is selected, and the \mathbf{X} is again imputed but under the original missing patterns $\mathcal{M}_0, \mathcal{M}_1$. This set of imputed values is now ready to be evaluated or used for downstream tasks.

The hyperparameters that we tune via this method are summarized in Table 15.2. In addition, we also use this cross-validation procedure to select the best method out of `opt.knn`, `opt.svm`, and `opt.tree`. We refer to this composite method as `opt.cv`. Similarly, we may use the cross-validation procedure for model selection for any set of imputations. We define `benchmark.cv` to be the procedure that selects the best method out of: `mean`, `pmm`, `bpc`, `knn`, and `iknn` that will be later used in computational comparisons (see Section 15.6 for descriptions of these individual methods).

Extensions to Multiple Imputation

Thus far, we have described `opt.impute` methods for single imputation which output a single completed data set. On the other hand, multiple imputation methods output $m \geq 2$ different completed data sets for a single missing data problem. Afterwards, analysis is performed on each of the m data sets separately, and the results are pooled [177]. For some applications, multiple imputation is preferred because it captures the variation in missing data imputation, which enables us to compute confidence intervals for downstream models trained on the imputed data sets. In addition, the pooled results from models fit on multiple imputed data sets may provide better point estimates than models fit on a single imputed data set in some cases.

To extend `opt.impute` to produce multiple imputations, we generate m warm starts using a probabilistic procedure, run `opt.knn`, `opt.svm`, or `opt.tree` from these starting points, and output the full set of m completed data sets. These warm starts can be generated from sample draws under a previously estimated posterior distribution; an example would be using outputs from the `mice` procedure [65] implemented using the `MICE` package in R. This provides us with a representative set of imputations found by the `opt.impute` algorithm, which converges to local optima. We refer to the multiple imputation method as `opt.mi`. In the computational experiments, we use the benchmark multiple imputation method `mice` to generate the warm starts.

Note that there are other possible ways of adapting `opt.impute` to the multiple imputation schema. We may introduce m instances of artificial noise in the observed values, and solve the resulting optimization problems. Alternatively, we may run `opt.impute` on m bootstrapped samples of the original data set. Afterwards, we can analyze each of the m imputed data sets separately and pool the results as before.

15.6 Real-World Data Experiments

In this section, we evaluate the performance of `opt.impute` on many real-world data sets. Our comparisons include 1) the effect on imputation accuracy, and 2) the effect on the performance of downstream machine learning tasks. We compare to the most commonly used state-of-the-art methods on a large sample of data sets from the UCI Machine Learning Repository. For data sets that include categorical variables, we impute the discrete values directly using our specialized imputation methods for categorical variables and benchmark methods.

Experimental Setup

To test the accuracy of the proposed missing data imputation method, we run a series of computational experiments on data sets taken from the UCI Machine Learning Repository for both regression and classification tasks. The data sets cover a range of number of observations n and number of features p , potentially mixed with both continuous and categorical variables. The numbers of continuous (p_0) and categorical (p_1) variables in each of these data sets are given in Table 15.9.

In these experiments, we use full data sets in which all entries are known, and we generate patterns of missing data for various percentages ranging from 10% to 50%. We take the full data sets \mathbf{X} that have no missing entries to be the ground truth. We run some of the most commonly-used and state-of-the-art methods for data imputation on these data sets to predict the missing values and compare against our optimization based imputation methods. The individual methods in this comparison are:

1. **Mean Impute** (`mean`): The simplest imputation method. For each missing value x_{id} , imputes the mean of all known values in dimension d .
2. **Predictive-Mean Matching** (`pmm`): An iterative method which imputes missing values from known values in a given dimension using linear regressions. It is commonly used for multiple imputation and can be generalized to multiple missing dimensions using the chained equations process [65]. Implemented using the `MICE` package in R.
3. **Bayesian PCA** (`bpc`): A missing data estimation method based on Bayesian principal component analysis [208]. Implemented using the `pcaMethods` package in R.
4. **K -Nearest Neighbors** (`knn`): A single-step, greedy method which imputes missing values using the K -nearest neighbors of an observation based upon Euclidean distance. The candidate neighbors must have non-missing values in the imputed feature. Averaged distance is used if some other coordinates are missing. Implemented using the `impute` package in R.
5. **Iterative K -Nearest Neighbors** (`iknn`): Implemented in R and Julia, based on the description in the original papers [52, 70].
6. **Optimal Impute** (`opt.impute`): All sub-methods below use warm starts including: `mean`, `knn`, `bpc` and five random starts where the values are imputed by a random sampling of the non-missing observations of that feature. The imputation which results in the lowest objective value is selected for each method.
 - (a) **K -NN based** (`opt.knn`): This method solves the optimal K -nearest neighbors problem (15.7). Convergence time depends

upon the quality of the initial warm start. We run both block coordinate descent and coordinate descent for small data sets of size $n \leq 10,000$, and only coordinate descent for large data sets with higher n . The implementation was in the programming language `Julia` with fast algorithms for K -nearest neighbor calculations.

- (b) SVM Regression and Classification based (`opt.svm`): This method solves the maximum margin support vector machine problem (15.11) using a radial basis function kernel. For continuous variables, we use ϵ -support vector regression; for categorical variables, we use classical support vector machines. These problems were solved using coordinate descent methods. The implementation was in `Julia` using the `scikit-learn` package in `Python`.
- (c) Decision Tree based (`opt.tree`): This method solves the optimal decision-tree problem (15.15). We use the OCT and ORT algorithms described in Chapters 8 and 10 for continuous and categorical variables, respectively.

In addition, we consider two composite methods: `opt.cv`, which selects the best method from `opt.knn`, `opt.svm`, and `opt.tree`; and `benchmark.cv`, which selects the best method from `mean`, `pmm`, `bpc`, `knn`, and `iknn`. These composite methods use the cross-validation procedure described in Section 15.5. To generate the validation set for each missing data problem, we randomly sample an additional 10% of the entries to be hidden under the MCAR assumption. After running each individual method, we select the one that gives the lowest MAE on the validation set. We re-run this method on the original missing data set to obtain the final imputation.

Each imputation method was run for a maximum time limit of 12 hours on each data set. The quality of the imputations is evaluated using the same MAE and RMSE metrics defined in Section 15.5. For each of the `opt.impute` methods, we also record and present the convergence in objective value and MAE to show the progress over the iterations.

Missing Pattern

Because the mechanism which generates the pattern of missing data can affect imputation quality, we run experiments under two different missing data mechanisms: missing completely at random (MCAR) and not missing at random (NMAR). These statistical assumptions are summarized in Table 15.3. The MCAR assumption implies that the missing pattern is completely independent from both the missing and observed values. The NMAR assumption implies that the missing pattern depends upon the missing values. There is an intermediate type of assumption, missing

Table 15.3: Statistical assumptions of mechanisms used to generate patterns of missing data \mathcal{M} for data set \mathbf{X} . Here, we suppose that f is the underlying density of the missing pattern, and $\mathbf{X}^{obs}, \mathbf{X}^{miss}$ are the observed and missing components of the data set, respectively.

Mechanism of Missing Data	Assumption
Missing Completely at Random (MCAR)	$f(\mathcal{M} \mathbf{X}^{obs}, \mathbf{X}^{miss}) = f(\mathcal{M})$
Missing at Random (MAR)	$f(\mathcal{M} \mathbf{X}^{obs}, \mathbf{X}^{miss}) = f(\mathcal{M} \mathbf{X}^{obs})$
Not Missing at Random (NMAR)	$f(\mathcal{M} \mathbf{X}^{obs}, \mathbf{X}^{miss})$ is a function of \mathbf{X}^{miss}

at random (MAR), which implies that the missing pattern depends only upon the observed values, but not upon the missing values. Because this assumption is less general than NMAR, we do not consider this mechanism for our experiments.

To generate MCAR patterns of missing data, we randomly sample a subset of the entries in \mathbf{X} to be missing, assuming that each entry is equally likely to be chosen. The NMAR patterns are generated by sampling missingness indicators as independent Bernoulli random variables where each probability p_{id} equals the probability that a normal random variable $N(x_{id}, \epsilon)$ is greater than a particular threshold for dimension d . The threshold for each dimension d is the quantile of \mathbf{X}^d which corresponds to the desired missing percentage level.

Note that regardless of the missing data scenarios generated for the experiments, in order to make fair comparisons, we always use MCAR as the generating mechanism for cross-validation.

Downstream Tasks

For 10 data sets from the UCI Machine Learning Repository, we run further experiments to evaluate the impact of these imputations on the intended downstream machine learning tasks. This selection includes a representative sample of 5 data sets for regression and 5 data sets for classification, with dependent variable observations $\mathbf{Y} \in \mathbb{R}^n$ and $\mathbf{Y} \in \{0, 1\}^n$ respectively. We evaluate both single and multiple imputation methods in these experiments.

For single imputation, we consider `opt.cv` and `benchmark.cv`. First, we divide each downstream data set using a 50% training/testing split. Next, we randomly sample a fixed percentage of the entries in \mathbf{X} to be missing completely at random, ranging from 10% to 50%. For each missing percentage, we impute the missing values in the training set and then fit standard machine learning algorithms to obtain a classification or regression model. We impute the missing values in the testing set by running the imputation methods on the full data set. For the regression tasks, we fit cross-validated Lasso and SVR models and compute the out-of-sample

accuracy on the imputed testing set. For the classification tasks, we fit cross-validated SVM and Optimal Trees models and compute the out-of-sample R^2 on the imputed testing set.

We also evaluate the performance of multiple imputation methods on the downstream tasks. In these experiments, we consider the following methods:

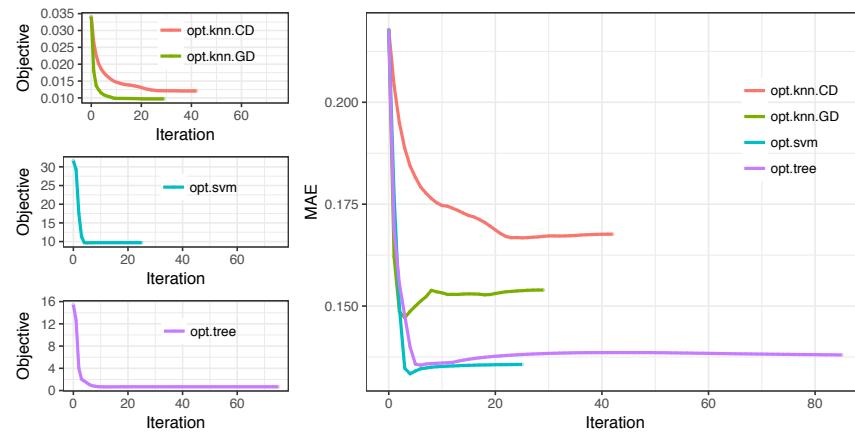
1. **Multivariate Imputation by Chained Equations** (`mice`): An iterative method which imputes each dimension with missing values one at a time drawing from distributions fully conditional on the other variables. We use predictive mean matching for continuous variables and logistic regression for categorical variables. This process is repeated to generate m fully imputed data sets. Implemented via the `MICE` package in R.
2. **Optimal Impute for Multiple Imputation** (`opt.mi`): Starting from m warm starts, we run `opt.knn`, `opt.svm`, or `opt.tree` to generate a new set of m fully imputed data sets. We use warm starts produced by `mice`, and the best model among K -NN, SVM, and trees is selected initially via cross-validation.

For both `mice` and `opt.mi`, we generate $m = 5$ multiple imputations for the training set and fit an ensemble of predictive models on these completed training sets. We make predictions on the test set by averaging the predictions from the model ensemble. For the classification tasks, we use a threshold value of 0.5. We run this experiment 100 times with different training/testing splits and distributions of missing values for each data set and report the averaged out-of-sample of the predictive models.

Results

We run the methods on 84 data sets from the UCI Machine Learning Repository. These data sets range in size from $n = 23$ to 5,875 observations and dimension $p = 2$ to 124. In the following sections, we first show the convergence for each of the `opt.impute` methods is fast and generally leads to a decrease in MAE. Next, we demonstrate that the quality of the imputations is significantly higher for `opt.impute` compared to the reference methods, and that this leads to improved performance on downstream classification and regression tasks. We further discuss the sensitivity of imputation quality to the model parameters (K , cp , C), warm starts, descent method (BCD or CD), and data characteristics including the missing pattern. Finally, we compare the computational burden of each method.

Figure 15.1: Solution paths of `opt.impute` methods on the `iris` data set. These plots show the objective value and mean absolute error (MAE) of the imputation over the course of the algorithm. Each path represents a different algorithm: `opt.knn` (BCD and CD), `opt.svm` (CD), and `opt.tree` (CD). Mean imputation warm start is used.



Convergence

Figure 15.1 represents the change in objective value and MAE over the iterations for each of the `opt.impute` methods based on mean warm start, using `iris` data set as an example. We present results for `opt.knn` (CD and BCD), `opt.svm` (CD), and `opt.tree` (CD). The convergence is relatively fast for all methods; in particular, the BCD algorithm for K -NN converges significantly faster than the CD algorithm. When comparing the change in MAE, the value generally monotonically decreases with each iteration in concordance with the change in objective, especially during the first few iterations. In some paths, MAE increases slightly after a certain point. RMSE exhibits the same behavior and is therefore not plotted. This suggests a potential issue of overfitting to the known observations, which may be remedied by regularization or early stopping. In summary, the solution paths illustrate: 1) convergence is often fast, and 2) the objective functions are decent proxies for out-of-sample MAEs, and 3) imputation quality for each first-order method generally improves until convergence.

In general, we found that the BCD algorithm for `opt.knn` did not significantly improve upon imputation accuracy compared to the CD algorithm, but only improved upon speed. Because the BCD algorithms do not scale as well, we restricted our analysis to the CD algorithms for `opt.svm` and `opt.tree`.

Imputation Accuracy

The imputation accuracy for each data set is presented in Table 15.9 for the scenario in which 30% of the entries are missing, assuming MCAR. We compare the benchmark ones and each individual `opt.impute` method (not cross-validated); the method with the lowest MAE (i.e., best imputation accuracy) is bolded. Among all data sets, at least one of the `opt.impute` methods obtains the lowest MAE in 76.2% of the data sets, followed by `iknn` and `bpcda` imputation methods with 9 and 4 wins each. Comparatively, `mean`, `knn`, and `pmm` impute have the weakest performances. Among the `opt.impute` methods, the tree based model achieves the lowest MAE in most data sets.

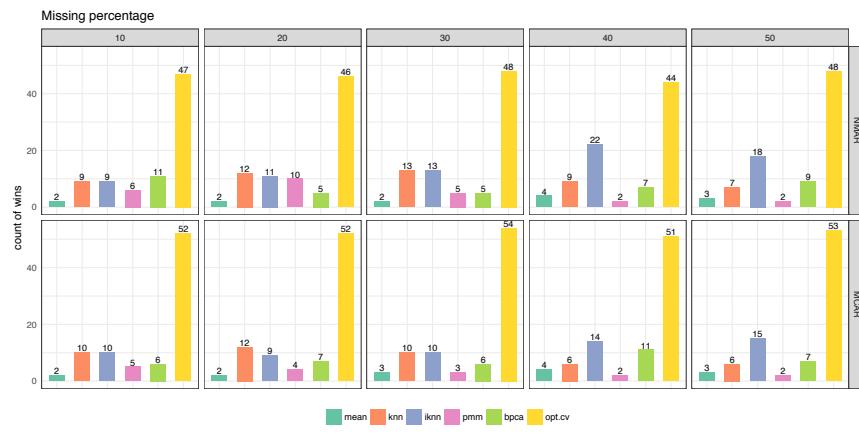
We repeat this experiment for other percentages of missing data with the winning counts summarized in Figure 15.2, using `opt.cv` as our proposed method. We show the number of times that each method achieves the best overall imputation with lowest MAE and RMSE under five different missing data percentages, as well MCAR and NMAR scenarios. In all missing data scenarios, our proposed method produces the best imputations in more than half of the data sets according to both performance metrics. Among the comparator methods, `mean` and `pmm` are generally among the weaker ones. When MAE is the metric, the heuristic method `iknn` performs the best among the benchmark methods, suggesting that the idea of iteratively updating the imputed values have merits. At higher percentages of missing values (the right-most subfigures), `bpcda` improves in its performance when RMSE is the metric of evaluation, but still not as strong as `opt.cv`.

In Figure 15.3, we present summary results of the MAE and RMSE values as geometric means across all data sets for each missing percentage and missing data mechanism, with the confidence bands representing one geometric standard deviation multiplied above and divided below by the mean. Comparatively, `opt.cv` achieves the lowest average MAE and RMSE values for all missing percentages. At the 10% missing data percentage, the average MAE of the `opt.cv` imputations is 0.100, a reduction of 14.9% from the average MAE of 0.118 obtained by the best benchmark method `knn`. As missing percentages increase, `opt.cv` remains the most accurate imputation method, with the average MAE of 0.142 at 50% missing, a reduction of 12.1% from the average MAE of 0.172 obtained by the next best method `knn`. The performance of `opt.cv` relative to benchmark ones does not appear to differ drastically between the MCAR and NMAR scenarios, with overall higher MAE for NMAR across most methods, as expected.

To isolate the effect of each individual method from the cross-validation procedure, we further summarize the results by comparing one method at a time against the benchmark ones. Table 15.4 presents the statistical comparisons between each `opt.impute` method and each benchmark method. We conduct pairwise Wilcoxon signed rank tests and

Figure 15.2: Number of data sets in which each missing data imputation method achieves lowest mean absolute error (MAE) or root mean squared error (RMSE) from true value, with ties included. Each panel represents a different missing percentage ranging from 10% to 50%. Panels in the top row are for not missing at random scenarios, whereas the ones in the bottom row are for missing completely at random scenarios.

(a) Counts based on lowest MAE



(b) Counts based on lowest RMSE

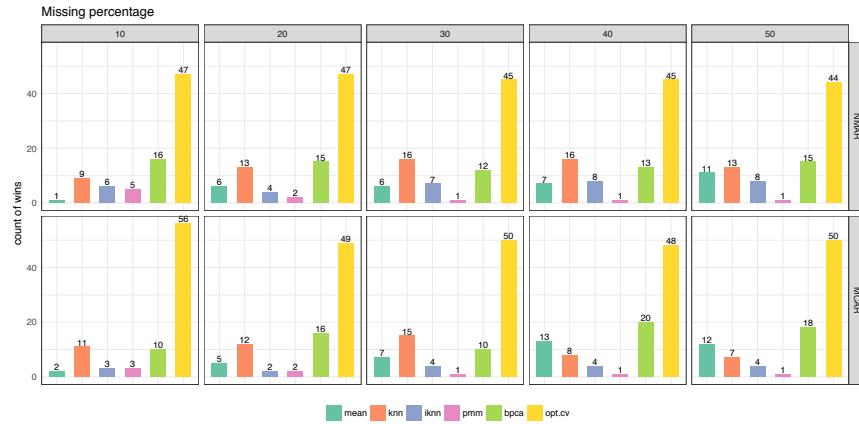


Table 15.4: Pairwise Wilcoxon signed-rank tests and t-tests between `opt.impute` and benchmark methods, with the p -values adjusted for multiple comparisons.

<code>opt.impute</code>	Benchmark	Δ rank (adjusted p -value)	Δ MAE (adjusted p -value)
opt.cv	mean	-0.7855 (<0.001***)	-0.0502 (<0.001***)
opt.cv	pmm	-0.8355 (<0.001***)	-0.0399 (<0.001***)
opt.cv	bpc	-0.6329 (<0.001***)	-0.0214 (0.0019**)
opt.cv	knn	-0.6281 (<0.001***)	-0.0134 (0.0499*)
opt.cv	iknn	-0.5352 (<0.001***)	-0.0199 (0.0046**)
opt.knn	mean	-0.6424 (<0.001***)	-0.0419 (<0.001***)
opt.knn	pmm	-0.6091 (<0.001***)	-0.0316 (<0.001***)
opt.knn	bpc	-0.4875 (<0.001***)	-0.0131 (0.0601)
opt.knn	knn	-0.3850 (<0.001***)	-0.0051 (0.4574)
opt.knn	iknn	-0.3611 (<0.001***)	-0.0116 (0.1011)
opt.svm	mean	-0.5852 (<0.001***)	-0.0355 (<0.001***)
opt.svm	pmm	-0.4875 (<0.001***)	-0.0252 (<0.001***)
opt.svm	bpc	-0.2515 (<0.001***)	-0.0067 (0.3335)
opt.svm	knn	-0.1371 (0.0033**)	+0.0013 (0.8485)
opt.svm	iknn	-0.0322 (0.0884)	-0.0052 (0.4589)
opt.tree	mean	-0.7139 (<0.001***)	-0.0454 (<0.001***)
opt.tree	pmm	-0.7712 (<0.001***)	-0.0351 (<0.001***)
opt.tree	bpc	-0.5137 (<0.001***)	-0.0165 (0.0176*)
opt.tree	knn	-0.4136 (<0.001***)	-0.0086 (0.2152)
opt.tree	iknn	-0.3135 (<0.001***)	-0.0151 (0.0337*)

paired t-tests between each pair of methods. When comparing `opt.cv` against the benchmark methods, our proposed cross-validated method achieves statistically significant lower rank and lower MAE compared to each benchmark. For each individual `opt.impute` method, with the exception of `opt.svm` against heuristic `iknn`, the `opt.impute` one has statistically significant lower rank than every benchmark. The decrease in MAE is still statistically significant when `mean`, `bpc`, and `pmm` are comparators, but no longer statistically significant when compared to `knn` or `iknn`. This suggests that each of the proposed methods holds its own against most benchmark ones, especially under rank comparisons, but the cross-validation procedure adds another layer of improvement in imputation quality.

Finally, we compare against the same cross-validated procedure introduced in Section 15.5 applied on all the benchmark methods (`benchmark.cv`) with results in Figure 15.2b. At 30% missing data, we observe 10.1% average improvement in MAE down to 0.118 from 0.131. Further, `opt.cv` achieves highest imputation accuracy in more than 78.6% of the data sets compared to `benchmark.cv`.

Performance on Downstream Tasks

Next, we evaluate the performance of standard machine learning algorithms for classification and regression trained on the imputed data. We consider

Figure 15.3: Mean absolute error (MAE) and root mean squared error (RMSE) across 84 data sets for each imputation method, comparing `opt.cv` against all benchmark methods and against the cross-validated best benchmark method, `benchmark.cv`. The center lines are geometric mean with one geometric standard deviation multiplied above and divided below. The x-axis corresponds to the percentage of missing entries.

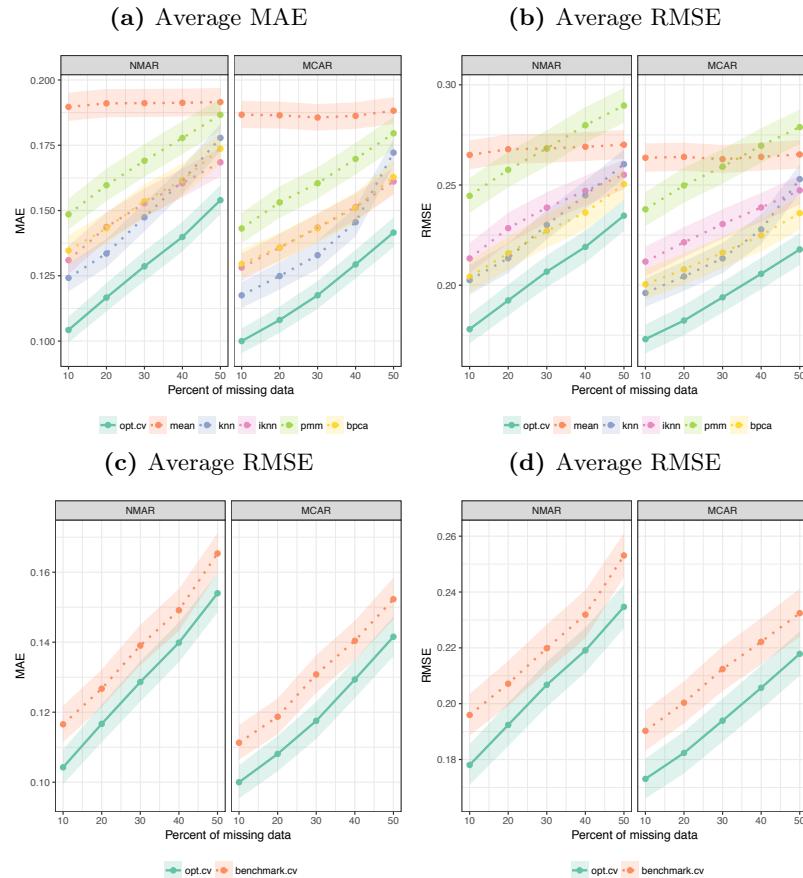


Table 15.5: Data sets considered for downstream regression and classification tasks. For classification tasks, we list the average baseline out-of-sample accuracy of an SVM model fit on the full data set, and for regression tasks, we list the average baseline out-of-sample R^2 of a Lasso model fit on the full data set.

Task	Name	(n, p)	Accuracy or R^2
Classification	climate-model-crashes	(540, 18)	0.95
	connectionist-bench	(990, 10)	0.93
	ecoli	(336, 8)	0.96
	iris	(150, 4)	1.00
	pima-indians-diabetes	(768, 8)	0.77
Regression	abalone	(4177, 7)	0.51
	auto-mpg	(392, 8)	0.82
	housing	(506, 13)	0.71
	parkinsons-telemonitoring-total	(5875, 16)	0.09
	wine-quality-white	(4898, 11)	0.27

the data sets in Table 15.5, which were selected as a representative subsample from the UCI Machine Learning Repository data sets. These data sets range in size, having $n = 150$ to 5,875 observations and $p = 4$ to 16 features. The difficulty of the regression or classification task on the completely known data set also varies widely. The baseline out-of-sample accuracy of an SVM model for the binary classification problems ranges from 77% to 100%, and the baseline out-of-sample R^2 of a LASSO model for the regression problems ranges from 0.09 to 0.82. For each of these data sets, the downstream tasks become more difficult as the missing data percentage increases.

In Figure 15.4, we show how the imputation method chosen impacts the performance for downstream tasks, across different data sets and different missing data percentages. In Tables 15.6 and 15.7, we show pairwise t-test results, aggregating out-of-sample performance results by downstream task and missing percentage. These results include comparisons for both single and multiple imputation methods.

For the single imputation methods, we observe that the improvement of `opt.cv` over the best cross-validated benchmark method is statistically significant for all missing percentages in both classification and regression tasks. Moreover, this improvement in out-of-sample accuracy and R^2 is monotonically increasing with the missing percentage. At 50% missing data, the average improvement in out-of-sample accuracy is 1.7% for classification tasks, and the average improvement in out-of-sample R^2 is 0.024 for regression tasks.

For the multiple imputation methods, we observe that the improvement of `opt.mi` over `mice` is statistically significant for all missing percentages in the regression tasks, and 3/5 missing percentages in the classification tasks. At the 50% missing percentage, the average

Table 15.6: Pairwise t-tests between `opt.impute` and benchmark methods for downstream classification tasks, with the p -values adjusted for multiple comparisons.

Missing %	Δ Out-of-Sample Accuracy (adjusted p -value)		
	<code>opt.mi - mice</code>	<code>opt.cv - benchmark.cv</code>	<code>opt.mi - opt.cv</code>
10	-0.0001 (1.0000)	0.0016 (0.0059**)	0.0006 (0.2076)
20	0.0018 (0.0059**)	0.0026 (<0.001***)	0.0008 (0.2076)
30	0.0005 (0.9858)	0.0082 (<0.001***)	0.0002 (1.0000)
40	0.0018 (0.0491*)	0.0113 (<0.001***)	0.0043 (<0.001***)
50	0.0052 (<0.001***)	0.0171 (<0.001***)	0.0038 (<0.001***)

Table 15.7: Pairwise t-tests between `opt.impute` and benchmark methods for downstream regression tasks, with the p -values adjusted for multiple comparisons.

Missing %	Δ Out-of-Sample R^2 (adjusted p -value)		
	<code>opt.mi - mice</code>	<code>opt.cv - benchmark.cv</code>	<code>opt.mi - opt.cv</code>
10	0.0014 (<0.001***)	0.0034 (<0.001***)	0.0013 (<0.001***)
20	0.0029 (<0.001***)	0.0113 (<0.001***)	0.0027 (<0.001***)
30	0.0071 (<0.001***)	0.0161 (<0.001***)	0.0077 (<0.001***)
40	0.0085 (<0.001***)	0.0195 (<0.001***)	0.0108 (<0.001***)
50	0.0097 (<0.001***)	0.0237 (<0.001***)	0.0174 (<0.001***)

improvement is 0.5% in out-of-sample accuracy for classification tasks and 0.010 in out-of-sample R^2 for regression tasks. While these improvements are smaller than those for single imputation, they are significant at the $p = 0.001$ level.

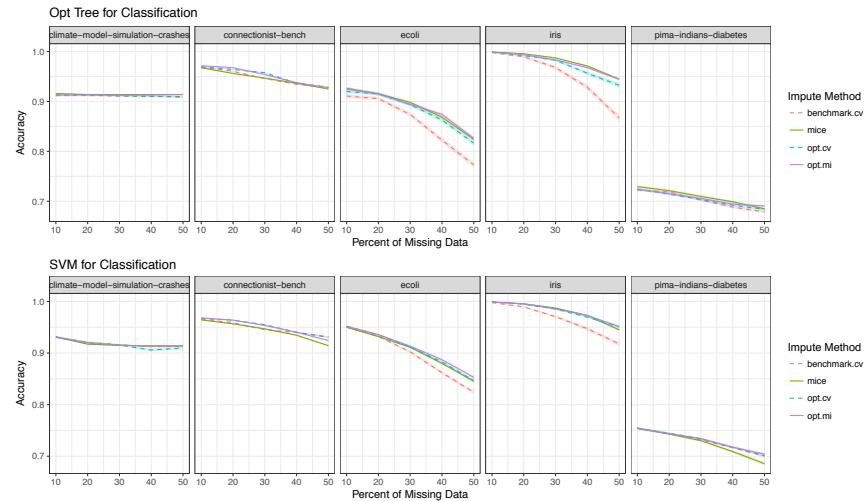
Overall, these results suggest that `opt.impute` leads to gains in out-of-sample performance in both single and multiple imputation settings. The relative improvements are consistently greatest at the highest missing percentages, where the imputation method selected has the largest impact on the downstream performance.

Finally, we compare the performance of single vs multiple imputation for `opt.impute`. We observe that the improvement of `opt.mi` over `opt.cv` is statistically significant in 8/10 scenarios, with the largest improvements occurring at the highest missing percentages. At the 50% missing percentage, the average improvement is 0.4% in out-of-sample accuracy for classification tasks and 0.017 in out-of-sample R^2 for regression tasks. These improvements are similar to the gains in performance over `mice`.

redo figure

Figure 15.4: Average out-of-sample performance of downstream models trained on data imputed via `opt.impute` and benchmark methods across a sample of classification and regression problems and a range of missing data percentages. Multiple and single imputation methods are solid and dotted lines respectively.

(a) Average out-of-sample accuracy values with standard errors of Optimal Trees and SVM models



(b) Average out-of-sample R^2 values with standard errors of SVR and Lasso models

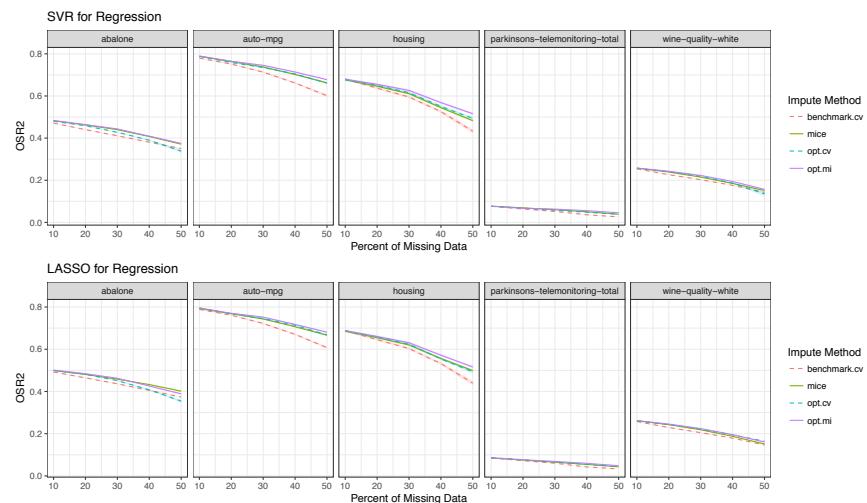
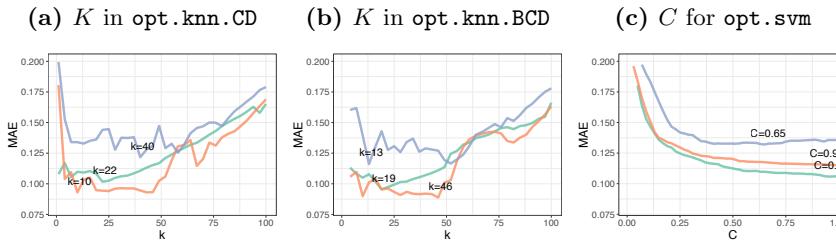


Figure 15.5: Sensitivity of MAE to the choice of K for the number of neighbors for K -NN coordinate descent, K -NN block coordinate descent, and the trade-off parameter C for SVM in data set `iris`. The colors represent different missing data percentages. The parameter value that achieves lowest MAE is labeled for each missing data percentage.



Sensitivity to Parameters

Model performance can be impacted by various parameters. For a specific data set and model, the performance can be sensitive to hyperparameters such as the number of neighbors K in K -NN and the trade-off parameter C for SVM. It is also affected by the number of random starts and choice of algorithm between block coordinate descent and coordinate descent. Data characteristics such as sample size n , feature dimension p , and missing data percentage may affect the imputation quality as well. This section explores how these parameters impact the imputation quality.

We found that all of the imputation model hyperparameters that we investigated affect imputation accuracy. Figure 15.5 shows the relationship between the hyperparameters and MAE for various data sets and missing patterns. For `opt.knn` (CD and BCD), the out-of-sample MAE first decreases and then increases as the hyperparameter increases. When K reaches the sample size, the imputation is equivalent to mean imputation. For `opt.svm`, the imputation accuracy remains relatively constant with respect to changes in parameter C after a certain threshold. There were no external parameters for trees, as the trees in each step were pruned during the training process. Overall, these plots suggest that the `opt.impute` methods are relatively robust even if their hyperparameters are not known exactly.

For `opt.knn`, the performances of block coordinate descent and coordinate descent are comparable. Under most missing data scenarios, block coordinate descent achieves the lower MAE in a few more data sets. As the missing data percentage increases, in many problems both block coordinate descent and coordinate descent methods find the same solutions, thus resulting in a tie. Comparing between the two, there is no clear dominant strategy; in practice we recommend running both methods and then selecting the imputation which yields the lowest objective value.

Table 15.8: Computational time comparison of benchmark and `opt.impute` imputation methods. Blank entries indicate that the method failed to converge with the 4 hour time limit.

Name	(n, p)	Missing %	Time (in seconds)						
			Benchmark			opt.impute			
			bpc	knn	pmm	knn.CD	knn.BCD	svm.CD	tree.CD
iris	(150, 4)	10	0.8	0.1	0.4	0.0	0.0	0.1	0.0
		30	1.7	0.4	0.5	0.0	0.0	0.5	0.1
		50	1.9	0.7	0.3	0.1	0.1	0.8	0.1
banknote	(1372, 4)	10	2.3	2.6	1.7	0.3	1.3	3.3	0.0
		30	14.1	14.9	1.9	0.8	5.0	15.6	0.1
		50	17.8	16.9	2.1	1.6	17.6	15.3	0.2
libras	(360, 90)	10	2.6	0.1	0.4	0.0	0.0	0.1	0.0
		30	3.4	0.4	0.5	0.0	0.0	0.5	0.1
		50	1.9	0.7	0.3	0.1	0.1	0.8	0.1
mushroom	(5644, 76)	10	26.4	387.4	4782.9	8.0	72.2	1442.9	-
		30	46.7	8.1	1068.5	12.8	17.6	-	-
		50	63.6	10.2	893.2	10.5	13.0	-	-
skin-seg.	(245057, 3)	10	392.3	1144.1	12193.1	1144.1	144.7	-	9.6
		30	450.6	1380.1	-	1420.6	-	-	15.6
		50	615.0	2503.5	-	2582.1	-	-	17.8
cnae-9	(1080, 856)	10	30.3	13.0	-	12.7	12.7	-	-
		30	58.2	14.0	-	13.9	14.0	-	-
		50	126.1	14.4	-	14.3	14.3	-	-

Computational Speed

Next, we compare the computational time required for all imputation methods across a selection of six UCI data sets and missing data patterns. Each method was run on a single thread of a machine with an Intel Xeon CPU E5-2650 (2.00 GHz) Processor and limited to 8 GB RAM with a time limit of 4 hours. For various `opt.impute` methods, we report the running times for `mean` warm starts, as multiple warm starts can be trivially parallelized. The results are shown below in Table 15.8.

Mean imputation is almost instantaneous and is therefore not presented in the table. For small-scale problems on the `iris` data set, all imputation methods finish quickly. As the data dimension p increases (for example, in the `libras-movement` data set), most `opt.impute` methods scale better than the `pmm` method. As the sample size n increases, `opt.knn.CD` also scales better than `pmm`, as seen in `banknote-authentication` and `skin-segmentation`. Among the `opt.impute` methods, tree based imputation scales very well with respect to sample size n but not dimension p . Despite its high imputation quality, SVM based imputation scales relatively poorly with respect to both n and p . Among the proposed methods, `opt.knn.CD` has the best scalability in both n and p .

In particular, when comparing coordinate descent and block coordinate descent methods, the former performs best when the data size is large. When n is in the 100,000s, the coordinate descent method still converges within one hour (see `skin-segmentation`). For the block coordinate descent method, each iteration requires solving a separate

system of linear equations for each continuous dimension, or an integer optimization problem for each of the categorical dimensions. On the other hand, the main bottleneck of `opt.knn.CD` is computing the K -NN assignment on \mathbf{X} to update \mathbf{Z} each iteration, which requires only $O(n \log n)$ time. When the problem size is small, the running times of the two methods are comparable, and the block coordinate descent method is slightly faster because it converges in fewer iterations. However, when the number of data entries to be imputed exceeds a certain threshold, the block coordinate descent method slows down and takes much longer. In practice, we recommend running both when $n \leq 10,000$ and performing model selection between the two, and running only coordinate descent when n is larger.

15.7 Concluding Remarks

We formulated in this chapter the missing data problem as a family of optimization problems. This framework accommodates almost any predictive model that describes the conditional relationship within the data, ranging from parametric to fully non-parametric models. By design, these formulations admit arbitrary missing pattern and mixed data types and do not require specific joint distributional assumptions on the data. In addition, we show how these methods can be used to generate multiple imputations.

The first-order methods that we developed to solve these optimization problem are highly scalable and produce high quality solutions. These methods are computationally fast; for example, the coordinate descent method for SVM solves problems with 100,000s of data points and 1,000s of features in seconds on a standard desktop computer. With more random starts, we obtain solutions which continue to improve upon the objective. Since random warm starts can be trivially parallelized, increasing the number of warm starts does not change the computational times materially if implemented efficiently.

For single imputation, we propose `opt.cv`, a combination method which uses cross-validation to select the best imputation objective function from K -NN, SVM, and decision tree models. We provide evidence on `opt.cv`'s strong empirical performance against benchmark single imputation methods in large scale computational experiments on 84 real-world data sets. For all of the missing data scenarios considered, `opt.cv` produces the best overall imputation for the largest number of data sets. In addition, `opt.cv` produces the lowest average MAE and RMSE for the majority of missing data scenarios. Our proposed cross-validation procedure generates additional missing pattern under MCAR, which may be further improved by adapting the generative procedure for more accurate reflection of imputation quality in the original data missing.

Further, we demonstrate that using the imputations produced by `opt.cv` with values closer to the ground truth leads to gains in out-of-sample performance on downstream regression and classification tasks. This suggests that at medium-to-high missing percentage scenarios, machine learning practitioners will benefit significantly by adopting this framework for single imputation.

For multiple imputation, we propose `opt.mi`, a method which runs `opt.impute` on a set of probabilistically generated warm starts. We show that this method offers a statistically significant improvement over both `mice` and `opt.cv` in the downstream tasks. However, the multiple imputation methods have drawbacks because they are computationally slower, require pooling after analyzing multiple data sets, and produce an ensemble of models which is less interpretable than a single model. Therefore, unless statistical inference is required, `opt.cv` may be preferable for many applications.

15.8 Notes and Sources

This chapter is from [43]. Table 15.10 depicts some of the state-of-the-art methods for missing data imputation.

The simplest method is mean impute, in which each missing value x_{id} is imputed as the mean of all observed values in dimension d . Mean impute underestimates the variance, ignores the correlation between the features, and thus often leads to poor imputation [177].

Joint modeling asserts some joint distribution on the entire data set. It assumes a parametric density function (e.g., multivariate normal) on the data given model parameters. In practice, model parameters are typically estimated using an Expectation-Maximization (EM) approach. It finds a solution (often non-optimal) of missing values and model parameters to maximize the likelihood function. Many software tools such as the R package `Amelia` 2 implement the EM method with bootstrapping, assuming that the data is drawn from a multivariate normal distribution [140]. Joint modeling provides useful theoretical properties but lacks the flexibility for processing data types seen in many real applications [263]. For example, when the data includes continuous and categorical variable types, standard multivariate density functions often fail at modeling the complexity of mixed data types. However, under the assumption that the categorical variables are independent, we can use mixture models of Gaussians and Multinomials for imputation [110].

In contrast to joint modeling, fully conditional specification is a more flexible alternative where one specifies the conditional model for each variable; it is especially useful in mixed data types [263]. To generalize to multivariate settings, a chained equation process — initializing using random sampling and conducting univariate imputations sequentially until

Table 15.9: Mean absolute errors of imputation methods on 84 data sets from the UCI Machine Learning repository with 30% missing values. The lowest MAE for each data set is indicated in bold.

Name	n	p_0	p_1	Benchmark					opt.impute		
				mean	pmm	bpcas	knn	iknn	knn	svm	tree
acute-1	120	1	5	0.370	0.363	0.231	0.269	0.360	0.229	0.227	0.219
acute-2	120	1	5	0.370	0.363	0.231	0.269	0.360	0.229	0.227	0.219
airfoil	1503	5	0	0.233	0.227	0.233	0.202	0.205	0.194	0.195	0.200
airline-costs	31	9	0	0.180	0.157	0.105	0.111	0.107	0.097	0.108	0.104
auto-mpg	392	5	2	0.240	0.179	0.154	0.162	0.169	0.140	0.136	0.129
balance-scale	625	4	0	0.301	0.411	0.301	0.350	0.311	0.370	0.320	0.305
banknote-auth.	1372	4	0	0.161	0.160	0.161	0.132	0.136	0.112	0.118	0.124
beer-aroma	23	8	0	0.204	0.200	0.177	0.177	0.184	0.173	0.164	0.163
blood-transfusion	748	4	0	0.112	0.122	0.112	0.095	0.088	0.080	0.082	0.066
breast-diagnostic	569	30	0	0.107	0.043	0.056	0.052	0.057	0.049	0.051	0.035
breast-prognostic	194	31	1	0.130	0.073	0.085	0.085	0.091	0.079	0.068	0.058
breast-cancer	683	8	1	0.246	0.153	0.132	0.154	0.179	0.137	0.136	0.133
climate-model	540	18	0	0.251	0.240	0.251	0.265	0.257	0.275	0.292	0.252
comm.-and-crime	129	99	23	0.161	0.290	0.140	0.098	0.125	0.097	0.094	0.071
comm.-and-crime-2	111	101	23	0.137	0.219	0.114	0.086	0.105	0.085	0.088	0.058
computer-hardware	209	7	0	1.109	1.180	0.109	0.093	0.109	0.092	0.178	0.093
concrete-compressive	103	7	0	0.234	0.200	0.206	0.205	0.198	0.185	0.187	0.175
concrete-flow	103	7	0	0.234	0.201	0.206	0.205	0.198	0.185	0.187	0.175
concrete-slump	103	7	0	0.234	0.201	0.206	0.205	0.198	0.185	0.187	0.175
congressional-voting	232	0	16	0.436	0.435	0.215	0.250	0.436	0.211	0.245	0.351
conn.-bench-sonar	208	60	0	0.163	0.121	0.144	0.109	0.122	0.107	0.092	0.091
conn.-bench	990	10	0	0.151	0.163	0.129	0.105	0.100	0.083	0.114	0.122
construction	33	4	0	0.361	0.246	0.364	0.330	0.284	0.328	0.325	0.398
contraceptive-method	1473	8	1	0.277	0.277	0.252	0.263	0.234	0.223	0.226	0.245
dermatology	358	33	1	0.225	0.145	0.145	0.121	0.142	0.108	0.136	0.196
diabetes	43	2	0	0.187	0.277	0.187	0.184	0.210	0.240	0.185	0.195
ecoli	336	7	0	0.122	0.122	0.094	0.107	0.091	0.099	0.111	0.090
fertility	100	7	2	0.353	0.385	0.343	0.343	0.348	0.337	0.345	0.367
flags	194	22	6	0.325	0.315	0.325	0.254	0.304	0.248	0.329	0.260
geographic-origin	1059	68	0	0.083	0.076	0.060	0.051	0.056	0.048	0.058	0.044
glass-identification	214	9	0	0.114	0.083	0.096	0.086	0.087	0.085	0.092	0.086
haberman-survival	306	3	0	0.170	0.226	0.170	0.175	0.166	0.173	0.173	0.170
hayes-roth	132	4	0	0.277	0.372	0.278	0.287	0.278	0.297	0.295	0.277
heart-disease	297	8	5	0.326	0.339	0.288	0.295	0.302	0.276	0.274	0.
hepatitis	80	4	15	0.309	0.302	0.309	0.275	0.263	0.257	0.266	0.348
hill-valley-noise	606	100	0	0.100	0.011	0.007	0.005	0.028	0.005	0.078	0.011
hill-valley	606	100	0	0.097	0.097	0.009	0.004	0.027	0.004	0.078	0.003
housing	506	13	0	0.182	0.121	0.115	0.099	0.104	0.080	0.105	0.126
hybrid-price	153	3	0	0.154	0.161	0.154	0.129	0.107	0.137	0.120	0.123
image-segmentation	210	19	0	0.145	0.081	0.086	0.064	0.067	0.063	0.085	0.063
immigrant-salaries	35	3	0	0.225	0.213	0.225	0.207	0.207	0.200	0.167	0.179
indian-liver-patient	579	8	2	0.104	0.095	0.094	0.098	0.087	0.091	0.097	0.079
ionosphere	351	34	0	0.202	0.174	0.155	0.111	0.119	0.117	0.121	0.148
iris	150	4	0	0.220	0.129	0.157	0.127	0.137	0.113	0.105	0.113
japan-emmigration	45	5	0	0.210	0.263	0.210	0.206	0.174	0.210	0.187	0.213
lenses	24	0	4	0.661	0.667	0.670	0.634	0.661	0.679	0.679	0.667
libras-movement	360	90	0	0.182	0.030	0.102	0.067	0.101	0.069	0.052	0.014
lpga-2008	157	6	0	0.146	0.177	0.142	0.145	0.141	0.150	0.129	0.130
lpga-2009	146	11	0	0.175	0.105	0.107	0.117	0.113	0.105	0.089	0.088
lung-cancer	27	0	56	0.368	0.348	0.364	0.343	0.368	0.359	0.335	0.344
mammographic	830	0	5	0.280	0.331	0.269	0.239	0.276	0.339	0.244	0.224
monks-problems-1	124	0	6	0.644	0.640	0.644	0.606	0.644	0.641	0.599	0.650
monks-problems-2	169	0	6	0.641	0.637	0.645	0.634	0.641	0.648	0.644	0.638
monks-problems-3	122	0	6	0.655	0.598	0.655	0.681	0.655	0.658	0.688	0.662
parkinsons-motor	5875	16	0	0.062	0.040	0.037	0.039	0.034	0.030	0.046	0.027
parkinsons-total	5875	16	0	0.062	0.040	0.037	0.039	0.034	0.030	0.046	0.027
parkinsons	195	21	0	0.135	0.089	0.085	0.075	0.081	0.069	0.082	0.069
pima-indians-diabetes	768	8	0	0.122	0.145	0.111	0.116	0.110	0.109	0.105	0.107
planning-relax	182	12	0	0.144	0.082	0.114	0.119	0.120	0.102	0.081	0.068
post-operative-patient	87	0	8	0.389	0.443	0.389	0.414	0.386	0.394	0.435	0.396
pyrim	74	27	0	0.180	0.124	0.176	0.117	0.119	0.115	0.122	0.128
qsar-biodegradation	1055	41	0	0.075	0.038	0.069	0.039	0.041	0.032	0.057	0.045
seeds	210	7	0	0.208	0.080	0.065	0.110	0.086	0.072	0.073	0.064
soybean-large	266	0	35	0.288	0.258	0.247	0.187	0.288	0.186	0.187	0.210
soybean-small	47	0	35	0.269	0.282	0.267	0.158	0.269	0.157	0.184	0.184
spect-f	80	0	22	0.217	0.213	0.208	0.184	0.214	0.195	0.187	0.191
spect-heart	80	44	0	0.114	0.163	0.131	0.123	0.120	0.114	0.114	0.114
statlog-satellite	4435	36	0	0.156	0.041	0.047	0.039	0.048	0.033	0.035	0.039
tele-dx-assistant	151	1	4	0.402	0.407	0.409	0.371	0.399	0.409	0.513	0.337
thoracic-surgery	470	3	13	0.147	0.170	0.139	0.143	0.146	0.142	0.221	0.140
thyroid-ann	3772	21	0	0.077	0.077	0.087	0.072	0.060	0.084	0.116	0.073
thyroid-new	215	5	0	0.094	0.108	0.089	0.085	0.075	0.077	0.089	0.085
triazines	186	60	0	0.157	0.067	0.118	0.050	0.071	0.045	0.089	0.050
tv-sales	31	8	0	0.207	0.195	0.181	0.193	0.173	0.195	0.173	0.196
vote-for-clinton	2704	9	0	0.064	0.072	0.054	0.068	0.055	0.052	0.063	0.054
wall-robot-2	5456	2	0	0.072	0.096	0.072	0.075	0.072	0.079	0.085	0.072
wall-robot-24	5456	4	0	0.092	0.117	0.092	0.089	0.087	0.086	0.095	0.090
wiki4he	176	0	44	0.220	0.223	0.186	0.187	0.197	0.178	0.173	0.209
wine-quality-red	1599	11	0	0.098	0.095	0.076	0.080	0.074	0.068	0.076	0.074
wine-quality-white	4898	11	0	0.076	0.078	0.067	0.077	0.065	0.060	0.068	0.060
wine	178	13	0	0.168	0.154	0.120	0.118	0.114	0.109	0.111	0.130
yacht	308	6	0	0.210	0.199	0.209	0.186	0.186	0.187	0.187	0.180
yeast	1484	8	0	0.072	0.092	0.069	0.074	0.067	0.068	0.093	0.068
zoo	101	1	15	0.289	0.283	0.184	0.152	0.286	0.150	0.364	0.148

Table 15.10: List of Imputation Methods

Method Name	Category	Software	Reference
Mean impute (<code>mean</code>)	Mean		[177]
Expectation-Maximization (<code>EM</code>)	EM		[81]
EM with Mixture of Gaussians and Multinomials	EM		[110]
EM with Bootstrapping	EM	<i>Amelia II</i>	[140]
<i>K</i> -Nearest Neighbors (<code>knn</code>)	<i>K</i> -NN	<code>impute</code>	[258]
Sequential <i>K</i> -Nearest Neighbors	<i>K</i> -NN		[162]
Iterative <i>K</i> -Nearest Neighbors	<i>K</i> -NN		[70, 52]
Support Vector Regression	SVR		[267]
Predictive-Mean Matching (<code>pmm</code>)	LS	<i>MICE</i>	[65]
Least Squares	LS		[48]
Sequential Regression Multivariate Imputation	LS		[225]
Local-Least Squares	LS		[161]
Sequential Local-Least Squares	LS		[280]
Iterative Local-Least Squares	LS		[67]
Sequential Regression Trees	Tree	<i>MICE</i>	[64]
Sequential Random Forest	Tree	<code>missForest</code>	[246]
Singular Value Decomposition	SVD		[258]
Bayesian Principal Component Analysis	SVD	<code>pcaMethods</code>	[208, 193]
Factor Analysis Model for Mixed Data	FA		[159]

convergence — is typically used [65]. Each iteration is a Gibbs sampler that draws from the conditional distribution on the imputed values.

A simple example of conditional specification is based on regression. Least-Squares (LS) imputation constructs single univariate regressions, regressing features with missing values on all of the other dimensions in the data. Each missing value x_{id} is then imputed as the weighted average of these regression predictions [48, 225]. Alternatively, in the Predictive-Mean Matching method (`pmm`), imputations are random samples drawn from a set of observed values close to regression predictions [65]. Imputation methods that use Support Vector Regression in place of LS for the regression step have also been explored [267].

When there is non-linear relationship between the variables, linear regression based imputation may perform poorly. [64] propose using Classification and Regression Trees (CART) as the conditional model for imputation. Extensions to random forests have also shown promising results [246]. These decision tree based imputation methods are non-parametric approaches that do not rely upon distributional assumptions on the data.

One of the most commonly used non-parametric approaches is *K*-Nearest Neighbors (*K*-NN) based imputation. This method imputes each missing entry x_{id} as the mean of the d th dimension of the *K*-nearest neighbors that have observed values in dimension d [258]. Some extensions of *K*-NN include sequential *K*-NN, which starts by imputing missing values from observations with the fewest missing dimensions and continues imputing the next unknown entries reusing the previously imputed values [162]. Iterative *K*-NN uses an iterative process to refine the estimates and choose the nearest neighbors based on the estimates from the previous iteration [70, 52]. The Local-Least Squares method

combines ideas from K -NN and LS, imputing each missing value x_{id} using regression models trained on the K -nearest neighbors of the point \mathbf{x}_i [161]. Sequential and iterative variations of Local-Least Squares resemble their K -NN imputation counterparts [280, 67].

Low dimensional representation-based imputation assumes that the data represents a noisy observation of a linear combination of a small set of principal components or factor variables. In the basic method, singular value decomposition (SVD) is used on the entire data set to determine the principal eigenvectors. The missing values are imputed as a linear combination of these eigenvectors. This process is iteratively repeated until convergence [258, 188]. Bayesian Principal Component Analysis is similar to SVD imputation but extends the method to incorporate information from a prior distribution on the model parameters [208, 193]. Some recent development of a variant of the EM algorithm for factor analysis also provides a missing data imputation method for mixed data [159].

Thus far, we have only discussed methods for single imputation which generate one set of completed data that will be used for further statistical analyses. Multiple imputation, on the other hand, imputes multiple times (each set is possibly different), runs the statistical analyses on each, and pools the results [177]. Such method is able to capture the variability in the missing data and therefore generate potentially more accurate estimates to the larger statistical problem. However, multiple imputation methods are slower and require pooling results, which may not be appropriate for certain applications.

Within the multiple imputation framework, the procedure for generating multiple estimates of missing values varies. Multivariate imputation by chained equations (`mice`), a popular multiple imputation method, generates estimates using: predictive mean matching, Bayesian linear regression, logistic regression, and others [65]. In all cases, the method initializes using random sampling and conducts univariate imputations sequentially until convergence. Each iteration is a Gibbs sampler that draws from the conditional distribution on the imputed values.

Chapter 16

The Power of Optimization Over Randomization

We should continually be striving to transform every art into a science: in the process, we advance the art.

— Donald Knuth

Contents

- 16.1. Limitations of Randomization
- 16.2. Optimization Approach
- 16.3. Optimization for Reducing Discrepancies
- 16.4. Optimization, Randomization, and Bias
- 16.5. Optimization for Reaching a Conclusion
- 16.6. Concluding Remarks
- 16.7. Notes and Sources

Experimentation on groups of subjects, similar in all ways but for the application of an experimental treatment, is a cornerstone of modern scientific inquiry. In any controlled experiment, the quality, interpretability, and validity of the measurements and inferences drawn depends upon the degree to which the groups are similar at the outset.

For close to a century, randomization of subjects into different groups has been relied upon to generate statistically equivalent groups. Where group size is large relative to variability, randomization robustly generates groups that are well-matched with respect to any statistic. However, when group sizes are small, the expected discrepancy in any covariate under randomization can be surprisingly large, hindering inference. This problem is further aggravated as the number of groups one needs to populate becomes larger.

This is the situation faced in numerous disciplines in which the rarity or expense of subjects makes assembly of large groups impractical. For example, in the field of oncology research, experimental chemotherapy agents are typically tested first in mouse models of cancer, in which tumor-bearing mice are segregated into groups and dosed with experimental compounds. Since these mouse models are laborious and expensive, group size is kept small (typically 8-10), while the number of groups is relatively large, to accommodate comparison of multiple compounds and doses with standard-of-care compounds and untreated control groups. In this case, it is clear that initial tumor weight is highly correlated with the post-treatment tumor weight, in which we measure the effect of treatment. A typical experiment might consist of 40-60 mice segregated into four to six groups of ten, though experiments using fewer mice per group and many more groups are performed as well. Given that the implanted tumors grow quite heterogeneously, a coefficient of variation of 50% or more in pre-treatment tumor size is not unusual.

In such circumstances, common in nearly all research using animal models of disease as well as many other endeavors, simple randomization fails to reliably generate statistically equivalent groups, and therefore fails to generate reliable inference. It is clearly more desirable that experiments be conducted with groups that are similar, in particular in mean and variance of relevant baseline covariates. Here we treat the composition of small statistically equivalent groups as a mathematical optimization problem in which the goal is to minimize the maximum difference in both mean and variance between any two groups. We report one treatment of this problem as well as a study of the size of the discrepancy when group enrollment is optimized compared to other common designs including complete randomization.

Block and orthogonal designs (see [97]) have been a common way to reduce variability when baseline covariates are categorical, but do not apply to mixed (discrete and continuous) covariates, which is the main focus of our work. For such cases, apart from randomization, two prominent

methods are pairwise matching for controlled trials (see [211] and [122]) and re-randomization as proposed in [194].¹ The finite selection model (FSM) proposed by [196] can also be used for this purpose. In comparisons explored in Section 16.3, we find that the balance produced by our proposed optimization-based approach greatly improves on both randomization and these methods.

Pairwise matching is most common in observational studies, where assignment to treatment cannot be controlled (see [232] and [229] for a thorough discussion of the application of pairwise matching and other methods to observational studies). A large impediment to the existing practices is that they are based on subject pairs. When sample sizes are small and random there will hardly be any well-matched pairs. We will see that such matching does little to eliminate bias in the statistics that measure the overall average effect size. Instead we consider matching the experimental groups in order to minimize the en-masse discrepancies in means and variances among groups as formulated in (16.1).

When discrepancy is minimized, statistics such as the mean difference in subject responses are far more precise, concentrated tightly around their nominal value, while still being unbiased estimates. Indeed, under optimization, these statistics will no longer follow their usual distributions, which are wider, and traditional tests that rely on knowledge of this distribution, like the Student T test, no longer apply. Beyond estimation, we propose a hypothesis test based on the bootstrap to draw inferences on the differences between treatments – inferences which experimental evidence shows are much more powerful than is usually possible.

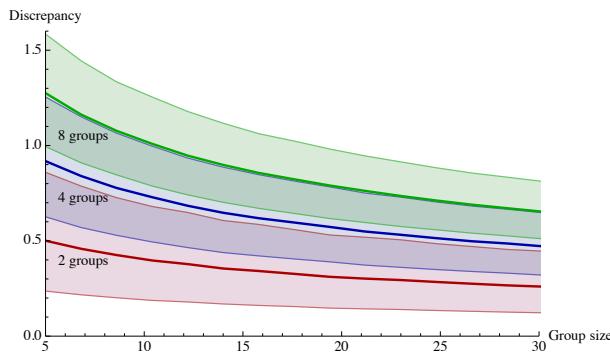
In this chapter, we provide theoretical and computational evidence that groups created by optimization have exponentially lower discrepancy in pre-treatment covariates than those created by randomization or by existing matching methods.

16.1 Limitations of Randomization

Three factors can impair successful matching of the independent variable means of groups assembled using randomization. These are: (a) the group size, (b) the variance of the data and (c) the number of groups being populated. The specific influence of these three factors is shown graphically in Figure 16.1. The plot shows the average maximal pair-wise discrepancy in means between groups under the conditions indicated for the normal distribution. Average discrepancy is proportional to standard deviation and is therefore reported in units of standard deviations.

¹The work of [194] can be seen as formalizing and reinterpreting the common informal practice of cherry-picking from several randomizations as a principled heuristic method for matching.

Figure 16.1: Average maximal pairwise discrepancy in means among randomly assigned groups of normal variates. The vertical axis is in units of standard deviation. The band denotes the average over- and under-shoot: $E[X|X \geq EX]$ and $E[X|X \leq EX]$ where X is maximal pairwise discrepancy.



It can be seen from the plot that discrepancy increases with the number of groups involved and decreases with increasing group size. When all three factors come into play: small group size, high standard deviation, and numerous groups, the degree of discrepancy can be substantial. For example, a researcher using randomization to create four groups of ten mice each will be left with an average discrepancy of 0.66 standard deviations between some two of the groups. Since statistical significance is often declared at a mean difference of 1.96 standard deviations ($p \leq 0.05$), this introduces enough noise into the experiment to conceal an effect in comparisons between the mismatched groups or to severely skew the apparent magnitude and statistical significance of a larger effect. Examination of Figure 16.1 makes it clear that when multiple groups are involved, even apparently large group size can still result in a substantial discrepancy in means between some groups. Doubling the group sizes to twenty each still leaves the researcher with a discrepancy of 0.47 standard deviations.

One solution to this problem is simply to increase group size until discrepancies decrease to acceptable levels. However, the size of the groups needed to do so can be surprisingly large. To reduce the expected discrepancy to below 0.1 standard deviations would require more than 400 subjects per group in the above experiment. For 0.01 standard deviations, more than 40,000 subjects per group would be necessary. With diminishing returns in the reduction of discrepancy with additional subjects, larger increases in the number of subjects enrolled are needed to conduct experiments studying subtler effects.

When considering the effects of this on post-treatment measurements such as mean differences or T statistic, it is clear that a more precise

measurement could be made when groups are well-matched at the onset. As we discuss below, well-matched groups yield a measurement that is much closer to the nominal (average or mode) measurement of pure randomization. Indeed, that this distribution of measurements is different (tighter) means that a naïve application of the Student T test would result in an underestimate of confidence and power, but that the distribution is tighter should allow for much more powerful inference.

16.2 Optimization Approach

Our proposal is to assign subjects so to minimize the discrepancies in centered first and second moments, where this assignment is gleaned via integer optimization. After assignment, we randomize which group is given which treatment, which ensures unbiased estimation as discussed in Section 16.4.

Given pre-treatment values of subjects w_i , $i = 1, \dots, n = mk$, we are interested in creating m groups each containing k subjects in such a way that the discrepancy in means and ρ times the discrepancy in second moments is minimized between any two groups. We first preprocess the full sample by normalizing it so that it has zero sample mean and unit sample variance. We set

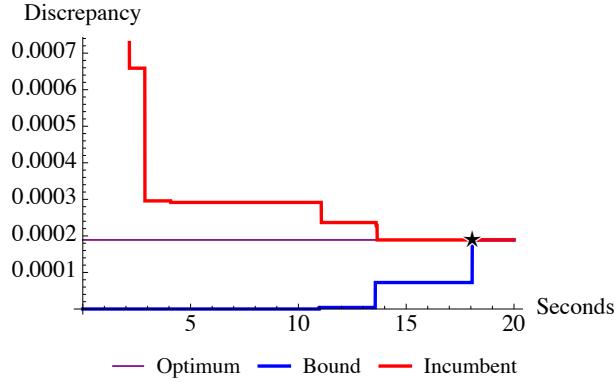
$$w'_i = (w_i - \hat{\mu})/\hat{\sigma}, \quad \text{where } \hat{\mu} = \sum_{i=1}^n w_i/n \quad \text{and} \quad \hat{\sigma}^2 = \sum_{i=1}^n (w_i - \hat{\mu})^2/n.$$

After construction of k groups, we randomize which treatment is given to which group. Algorithmically, we number the treatments and the groups in any way, shuffle the numbers $1, \dots, m$ and treat the group in position j with treatment number j . This does not affect the objective value.

The parameter ρ controls the tradeoff between the discrepancy of first moments and of second moments and is chosen by the researcher. We introduce the decision variable $x_{ip} = 0$ or 1 to denote the assignment of subject i to group p . Using a continuous auxilliary variable d and letting

$$\mu_p(x) = \frac{1}{k} \sum_{i=1}^n w'_i x_{ip} \quad \text{and} \quad \sigma_p^2(x) = \frac{1}{k} \sum_{i=1}^n w'^2_i x_{ip},$$

Figure 16.2: The progress of solving an instance of problem (16.1) with $n = 40, m = 4$.



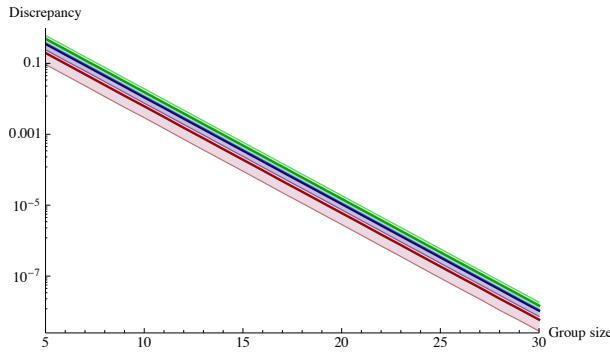
we formulate the problem as follows:

$$\begin{aligned}
 Z_m^{\text{opt}}(\rho) &= \min_x \max_{p \neq q} |\mu_p(x) - \mu_q(x)| + \rho|\sigma_p^2(x) - \sigma_q^2(x)| \\
 &= \min_{x,d} d \\
 \text{s.t. } \forall p < q = 1, \dots, m : \\
 &\quad d \geq \mu_p(x) - \mu_q(x) + \rho\sigma_p^2(x) - \rho\sigma_q^2(x) \\
 &\quad d \geq \mu_p(x) - \mu_q(x) + \rho\sigma_q^2(x) - \rho\sigma_p^2(x) \\
 &\quad d \geq \mu_q(x) - \mu_p(x) + \rho\sigma_p^2(x) - \rho\sigma_q^2(x) \\
 &\quad d \geq \mu_q(x) - \mu_p(x) + \rho\sigma_q^2(x) - \rho\sigma_p^2(x) \\
 &\quad x_{ip} \in \{0, 1\} \\
 &\quad \sum_{i=1}^n x_{ip} = k \quad \forall p \in [m] \\
 &\quad \sum_{p=1}^m x_{ip} = 1 \quad \forall i \in [n] \\
 &\quad x_{ip} = 0 \quad \forall i < p.
 \end{aligned} \tag{16.1}$$

As formulated, problem (16.1) is a mixed integer linear optimization problem with $m(1 + 2n - m)/2$ binary variables and 1 continuous variable. The last constraint reduces the redundancy due to permutation symmetry. Further symmetry reduction in the branch-and-bound process is possible by methods described in [152], which we included in our implementation. This symmetry is reintroduced by randomizing which group received which treatment.

We implement this optimization model in IBM ILOG CPLEX v12.5.

Figure 16.3: Discrepancy in means among optimally assigned groups of normal variates with $\rho = 0$. The colors are as in Figure 16.1. Note the vertical log scale compared to the absolute scale of Figure 16.1.



For values $n = 40$ and $m = 4$ problem (16.1) can be solved to full optimality in under twenty seconds on a personal computer with 8 processor cores. We plot the progress of the branch and bound procedure for one example in Figure 16.2. For larger instances, CPLEX generally finds a solution with objective value that is near optimal within a few minutes, while finding the optimum can take longer and proving its optimality even longer.

The formulation of optimization problem (16.1) extends to multiple covariates. Suppose we are interested in matching the first and second moments in a vector of r covariates where w_{is} denotes the s^{th} covariate of subject i . Again, we normalize the sample to have zero sample mean and identity sample covariance by setting $\mathbf{w}'_i = \Gamma\mathbf{w}_i - \hat{\mu}$, where Γ is the matrix square root of the (pseudo-)inverse of the sample covariance $\hat{\Sigma} = \sum_{i=1}^n \mathbf{w}_i - \hat{\mu}\mathbf{w}_i - \hat{\mu}^T/n$. Given the tradeoff parameter ρ , we rewrite the optimization problem for this case using $m(1 + 2n - m)/2$ binary variables and $1 + m(m - 1)r(r + 3)/4$ continuous variables as follows:

$$\begin{aligned}
 & \min d \\
 \text{s.t. } & x \in \{0, 1\}^{n \times m}, x_{ip} = 0 \quad \forall i < p, d \geq 0 \\
 & \sum_{i=1}^n x_{ip} = k \quad \forall p = 1, \dots, m \\
 & \sum_{p=1}^m x_{ip} = 1 \quad \forall i = 1, \dots, n \\
 & x_{ip} = 0 \quad \forall i < p \\
 & M \in \mathbb{R}^{\frac{m(m-1)}{2} \times r}, V \in \mathbb{R}^{\frac{m(m-1)}{2} \times \frac{r(r+1)}{2}} \\
 & \forall p = 1, \dots, m, q = p + 1, \dots, m :
 \end{aligned}$$

$$\begin{aligned}
d &\geq \sum_{s=1}^r M_{pq_s} + \rho \sum_{s=1}^r V_{pq_ss} + 2\rho \sum_{s=1}^r \sum_{s'=s+1}^r V_{pq_ss'} \\
\forall s = 1, \dots, r : \\
M_{pq_s} &\geq \frac{1}{k} \sum_{i=1}^n w'_{is} x_{ip} - x_{iq} \\
M_{pq_s} &\geq \frac{1}{k} \sum_{i=1}^n w'_{is} x_{iq} - x_{ip} \\
\forall s = 1, \dots, r, s' = s, \dots, r : \\
V_{pq_ss'} &\geq \frac{1}{k} \sum_{i=1}^n w'_{is} w'_{is'} x_{ip} - x_{iq} \\
V_{pq_ss'} &\geq \frac{1}{k} \sum_{i=1}^n w'_{is} w'_{is'} x_{iq} - x_{ip}.
\end{aligned}$$

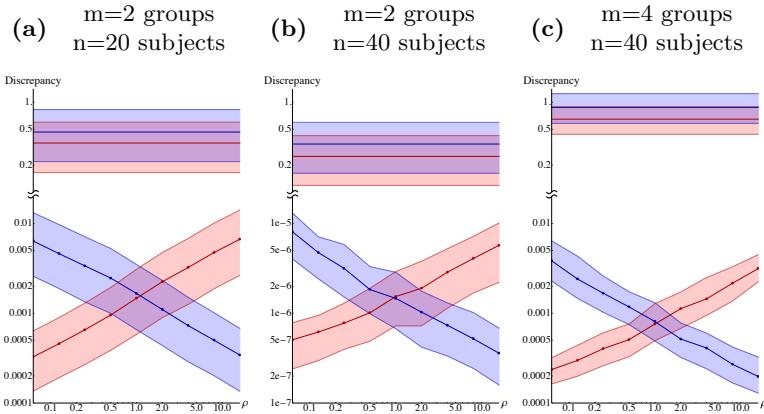
The potential extension to even higher moments is straightforward.

16.3 Optimization for Reducing Discrepancies

Using the above optimization model implemented in IBM ILOG CPLEX v12.5, we conducted a series of simulations comparing the results of group assembly using randomization and optimization. Our key finding is that optimization is *starkly* superior to randomization in matching group means under all circumstances tested.

Figure 16.3 provides the analogue of Figure 16.1 for optimization and Figure 16.4 compares side-by-side the mismatch achieved in the first two moments by optimization and by randomization. In particular we show for various numbers of groups and group sizes the achievable range of feasible matchings as ρ varies. For all values of ρ , the pre-treatment discrepancy is significantly reduced compared to that seen under randomization, essentially eliminating population variance as a significant source of noise for all but the most extreme circumstances. Noting that discrepancy in either moment is minuscule under optimization using any of the values of ρ shown, we arbitrarily choose $\rho = 0.5$ for all further numerical examples unless otherwise noted. To revisit the example used to illustrate the limitations of randomization, the researcher assembling four groups of ten mice each under optimization with $\rho = 0.5$ would end up with 0.0005 standard deviations of discrepancy in first moment (or a twentieth of that for $\rho = 0$, not shown in figure), compared with 0.66 standard deviations under randomization.

Figure 16.4: The range of achievable discrepancies under optimization and under randomization. The upper halves of the plots correspond to randomization and the lower ones to optimization. Red denotes discrepancy in mean and blue variance. The bands depict average under- and over-shoot. Notice the log scales and the break in the vertical axis.



There is some theoretical backing to the experimental evidence that optimization eliminates all discrepancies to such an extreme degree. When $\rho = 0$ and $m = 2$ the problem, scaled by $1/n$, reduces to the well-studied balanced number partitioning problem (see [154]). Let Z_2^{rand} denote the discrepancy in means under randomization. When pre-treatment covariates are random with variance σ^2 , we have by Jensen's inequality that

$$E[Z_2^{\text{rand}}] \leq \sqrt{E[(Z_2^{\text{rand}})^2]} = \sqrt{\frac{2}{k}}\sigma$$

and if they are normally distributed then

$$E[Z_2^{\text{rand}}] = \frac{2}{\sqrt{\pi k}}\sigma.$$

In comparison, an analysis of balanced number partitioning with random weights (see [155]) yields that there is a $C > 0$ such that

$$\text{median}Z_2^{\text{opt}}(0) \leq \frac{C}{2^{2k}}$$

and heuristic arguments from spin-glass theory (see [190]) provide the prediction

$$E[Z_2^{\text{opt}}(0)] = \frac{2\pi\sigma}{2^k},$$

Table 16.1: The number of subjects per group needed to guarantee an expected discrepancy no more than $\epsilon\sigma$ for $m = 2$ and $\rho = 0$.

ϵ	k^{Opt}	k^{Rand} ($\approx k^{\text{FSM}}$)	k^{Pair}	k^{RR}
0.1	3	128	9	4
0.01	5	12833	65	83
0.001	7	1273240	514	8130
0.0001	8	127323955	4354	820143

which agrees with our experimental results for large k . Comparing the asymptotic orders of Z_2^{rand} and $Z_2^{\text{opt}}(0)$, we see an *exponential reduction* in discrepancies by optimization versus randomization.

Matching done on a subject-pair-wise basis such as caliper matching as done in propensity score matching (see [232]) does not close this gap either even when the sample-based optimal caliper width is chosen. Consider for simplicity uniformly-distributed pre-treatment covariates so that any subsequent difference of two nearest neighbors are on average $n + 1^{-1}$. If assignment within each pair is randomized independently a simple calculation then shows that the average discrepancy is of order $k^{-3/2}$ whereas if assignment is alternating among the sorted covariates then the average discrepancy is of order k^{-1} . The case is worse for normally-distributed covariates as reported below.

Following the average predictions for the normal distribution, if we want to limit discrepancy to some fraction of the standard deviation, $\epsilon\sigma$, we see a dramatic difference in the necessary number of subjects per group, k :

$$k^{\text{Opt}} = \left\lceil \log_2 \frac{2\pi}{\epsilon} \right\rceil, \quad k^{\text{Rand}} = \left\lceil \frac{4}{\pi\epsilon^2} \right\rceil.$$

In Table 16.1 we report specific values of k^{Opt} and k^{Rand} , as well as k^{PW} corresponding to optimal pairwise matching and k^{RR} corresponding to the Mahalanobis-distance re-randomization method of [194] with a fixed acceptance probability of 5%.² This is a clear example of the power of optimization for experiments hindered by small samples. While pairwise matching and re-randomization improve upon randomization, they are significantly outperformed by optimization especially when small discrepancy is desired.

A concern may be that by optimizing only the first two moments and not others those higher moments may become mismatched. We find, however, that this is not the case even when compared to all the other methods considered above. In Table 16.2 we tabulate the mismatch

²Simulation is used to glean k^{opt} for these values of ϵ , for which the asymptotic predictions yield overestimates. Simulation also shows that for FSM, $k^{\text{FSM}} \approx k^{\text{Rand}}$.

Table 16.2: The discrepancy in various moments under different assignment mechanisms. Column ℓ corresponds to the average mismatch in the ℓ^{th} moments between the two groups and the last column corresponds to the mismatch in the generalized moments in $\log |w|$.

k	Method	1	2	3	4	5	\log
5	Opt	0.051	0.286	1.43	2.67	9.75	0.498
	Rand	0.510	0.689	1.79	3.81	10.30	0.544
	Pair	0.184	0.498	1.27	3.29	8.93	0.345
	Re-rand	0.047	0.711	1.09	3.88	8.47	0.572
	FSM	0.508	0.553	1.76	3.33	10.20	0.440
10	Opt	0.002	0.015	0.91	1.47	6.87	0.338
	Rand	0.352	0.504	1.30	2.88	7.79	0.399
	Pair	0.084	0.259	0.76	2.09	6.06	0.176
	Re-rand	0.030	0.497	0.76	2.93	6.20	0.389
	FSM	0.374	0.334	1.33	2.26	7.90	0.264
20	Opt	0.000	0.000	0.600	1.04	5.23	0.221
	Rand	0.258	0.345	0.947	2.13	6.13	0.276
	Pair	0.038	0.140	0.445	1.40	4.24	0.286
	Re-rand	0.021	0.356	0.565	2.16	4.99	0.284
	FSM	0.249	0.190	0.896	1.50	5.89	0.146

in the first five moments and in the generalized moment of \log for the various methods when assigning $2k$ subjects with baseline covariates drawn from a standard normal population. In Table 16.3 we tabulate the mismatch of multivariate moments for the various methods when assigning $2k$ subjects with multivariate baseline covariates drawn from a three-dimensional standard normal population. For pairwise matching we use the Mahalanobis pairwise distance, for re-randomization we use an acceptance probability of 5%, for FSM we use the method implied by equation (2.11) of [196] with $c_i = 1$, $T = I$, and for our method we use $\rho = 0.5$. We notice that optimal assignment yields superior balance in the moments considered and that all methods result in similar balance for those moments not directly considered in the optimization problem.

16.4 Optimization, Randomization, and Bias

Randomization has traditionally been used to address two kinds of bias in experimental design. The first is investigator bias, or the possibility that an investigator may subconsciously or consciously construct experimental groups in a manner that biases toward achieving a particular result. As

Table 16.3: The discrepancy in various multivariate moments under different assignment mechanisms. Column w_1w_2 , for example, corresponds to the average mismatch in the moments of w_1w_2 between the two groups, which by symmetry is the same as that of w_1w_3 or w_2w_3 on average.

k	Method	w_1	w_1^2	w_1w_2	w_1^3	$w_1^2w_2$	$w_1w_2w_3$
10	Opt	0.070	0.145	0.183	0.93	0.508	0.337
	Rand	0.360	0.492	0.344	1.29	0.580	0.333
	Pair	0.179	0.383	0.271	0.96	0.478	0.299
	Re-rand	0.141	0.493	0.357	0.88	0.484	0.340
	FSM	0.368	0.606	0.503	1.30	0.574	0.340
15	Opt	0.023	0.045	0.117	0.72	0.411	0.292
	Rand	0.292	0.400	0.286	1.05	0.489	0.289
	Pair	0.125	0.290	0.201	0.75	0.380	0.247
	Re-rand	0.113	0.409	0.289	0.71	0.414	0.293
	FSM	0.289	0.597	0.491	1.05	0.488	0.281
25	Opt	0.003	0.005	0.078	0.547	0.315	0.227
	Rand	0.226	0.325	0.222	0.842	0.384	0.227
	Pair	0.085	0.196	0.143	0.547	0.276	0.172
	Re-rand	0.086	0.326	0.230	0.566	0.314	0.220
	FSM	0.219	0.592	0.494	0.823	0.388	0.224

a fixed, mechanical process, optimization guards against this possibility at least as well as randomization. Indeed it does better because any manual manipulation of the optimized results would make the result less well-matched than the reproducible optimum, which is checkable, whereas no one grouping can ever be verified to truly be the result of pure randomization.

The second sort of bias is the incidental disproportionate assignment of variables, measured or hidden, that directly affect the treatment. Randomization, given large enough samples, will tend to equalize the apportionment of any one factor. However, just as with the measured covariates w_i , randomization cannot be counted upon to eliminate discrepancies in hidden factors when samples are relatively small. Optimization considers the measured covariates w_i when allocating a subject to a particular group. For all factors that are independent with this variable, the allocation remains just as random. Variables that are correlated with the measured covariates in ways such as joint normality will be just as well balanced as the measured covariates and variables with a higher order dependence, such as having a polynomial conditional expectation in w , would be as balanced as seen in Tables 16.2 and 16.3.

In general, as in a randomized experiment, the observed difference

in treatment effects will always be an *unbiased estimator* of the true population average difference. This is a consequence of randomizing the identity of treatments so that the assignment of a single subject is marginally independent of its potential responses to different treatments.³ Unbiasedness in estimation means that were the experiment to be repeated many times and the results recorded, the average result would coincide with the true value. In particular, there is no omitted variable bias. That is, neglecting to take into consideration a relevant covariate does not introduce bias in estimation.

16.5 Optimization for Reaching a Conclusion

As we have shown in the previous sections, optimization eliminates nearly all noise due to pre-treatment covariates. One would then expect that it can also offer superior precision in estimating the differences between treatments and superior power in making statistical inferences on these differences.

In randomized trials, randomization tests (see [237]) can be used to draw inferences based directly on the randomness of assignment without normality assumptions, which often fail for small samples. However, for optimization the assignment is not random and this is not applicable. The randomization test can, however, be supplanted by a bootstrap test.

Comparing between two treatments, we would like to test the null hypothesis that every subject $i = 1, \dots, n$ would have had the same response to treatment whether either of the two treatments were assigned. Let v_i denote the response measured for subject i after it was administered the treatment to which it was assigned. Then we propose the following test based on the bootstrap (see [89]). Given subjects with covariates w_1, \dots, w_n :

1. Find an optimal assignment of these to two groups (permuting randomly):
 $\{i_1, \dots, i_{n/2}\}$ and $\{i_{n/2+1}, \dots, i_n\}$.
2. Administer treatments and measure responses v_i .
3. Compute $\delta = \frac{1}{k}v_{i_1} + \dots + v_{i_{n/2}} - \frac{1}{k}v_{i_{n/2+1}} - \dots - v_{i_n}$.
4. For $b = 1, \dots, B$:
 - (a) Draw a random sample with replacement $w_{b,1}, \dots, w_{b,n}$ from w_1, \dots, w_n .

³The correctness of modeling using potential outcomes is contingent on the stable unit treatment value assumption. See [233].

- (b) Find an optimal assignment of these to two groups (permuting randomly):

$$\{i_{b,1}, \dots, i_{b,n/2}\} \text{ and } \{i_{b,n/2+1}, \dots, i_{b,n}\}.$$

$$(c) \text{ Compute } \delta_b = \frac{1}{k} v_{i_{b,1}} + \dots + v_{i_{b,n/2}} - \frac{1}{k} v_{i_{b,n/2+1}} + \dots + v_{i_{b,n}}.$$

$$5. \text{ Compute the } p\text{-value } p = \frac{1}{1+B} \left(1 + \sum_{b=1}^B \mathbb{I}[|\delta_b| \geq |\delta|] \right).$$

Then, to test our null hypothesis at a significance of α , we only reject it if $p \leq \alpha$. The quantity δ above constitutes our estimate of the difference between the two treatments.

To examine the effect of optimization on making a conclusion about the treatments we consider again the example of a murine tumor study. We consider two groups, each of k mice, with tumor weights initially normally distributed with mean 200mg and standard deviation 300mg (truncated to be nonnegative). Two treatments are considered: a placebo and a proposed treatment. Their effect on the tumor, allowed to grow for a period of a day, is of interest to the study.

The effects of treatment and placebo are unknown and are to be inferred from the experiment. We consider a hidden reality where the growth of the tumors are dictated by the Gomp-ex model of tumor growth (see [270]). That is, growth is governed by the differential equation:

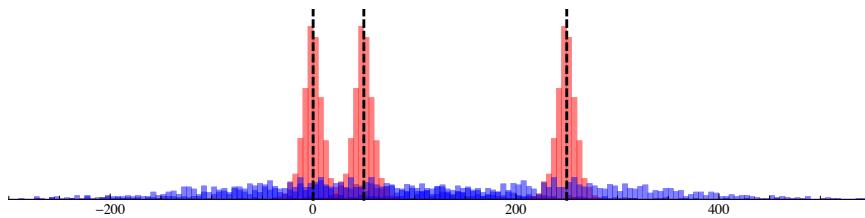
$$\frac{dw}{dt} = w(t)(a + \max\{0, b \log(w_c/w(t))\}),$$

where a and b are rate parameters and w_c is the critical weight that marks the change between exponential and logistic growth. We arbitrarily choose $a = 1 \frac{1}{\text{day}}$, $b = 5 \frac{1}{\text{day}}$, $w_c = 400\text{mg}$, and $t = 1\text{day}$. We pretend that tumors under either treatment grow according to this equation but subtract δ_0 from the final weights for the proposed treatment. We consider δ_0 being 0mg (no effect), 50mg (small effect), and 250mg (large effect).

For various values of k and for several draws of initial weights, we consider assignments produced by randomization, our optimization approach ($\rho = 0.5$), pairwise matching, and re-randomization. We consider both the post-treatment estimate of the effect and the inference drawn on it at a significance of $\alpha = 0.05$, using our bootstrap test for our method and the standard randomization test for the others.⁴ In Figure 16.5 we plot the resulting estimates for $k = 20$ and in Figure 16.6 we plot the rates at which the null hypothesis is rejected. When there is no effect, this rate should be no more than the significance $\alpha = 0.05$. When there is an effect, we would want the rate to be as close to 1 as possible. In a sense, the complement of this rate is the fraction of experiments squandered in pursuit of an effective drug. The cost-saving benefits of optimization in this case are clear.

⁴For non-completely-randomized designs, the randomization test draws random re-assignments according to the method employed at the onset. See Chapter 10 of [237].

Figure 16.5: The distribution of estimates of effect size under optimization (red) and randomization (blue) for $k = 20$ and effect sizes 0mg, 50mg, and 250mg (dashed lines). The overlap of estimates under randomization of the nonzero effects and of the zero effect elucidate the low statistical power of randomization in detecting the nonzero effects.

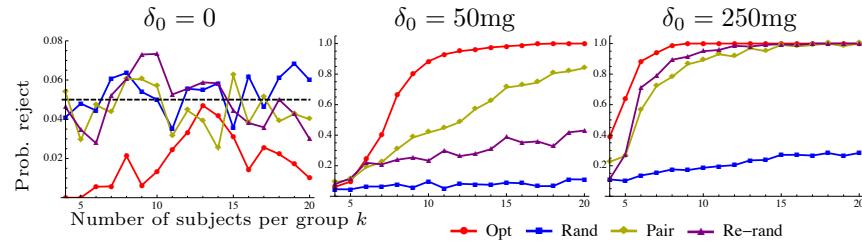


The exact improvements in precision and power depend on the nature of treatment effect. However, comparisons to the existing methods are possible. [194] study reduction in variance due to re-randomization only under the additive treatment model, a very restrictive assumption. In this setting, when setting $\rho = 0$, the same analysis as provided in their Theorem 3.2 provides that the reduction in variance provided by en-masse optimization is exponentially better because the reduction in mean mismatch is exponentially better. Nonetheless, treatment effects usually do depend, albeit perhaps to a lesser extent, on higher orders of the covariates and on their interactions. In Tables 16.2 and 16.3 we saw that optimization balances higher and interaction moments no worse than other methods (better for second moments).

16.6 Concluding Remarks

In this chapter, we presented evidence that optimization produces groups that are far more similar in mean and variance than those created by

Figure 16.6: The probability of rejecting the null hypothesis of no effect for various effect sizes.



randomization, especially in situations in which group size is small, data variability is large, and numerous groups are needed for a single experiment. For each additional subject per group, optimization roughly halves the discrepancy in the covariate, whereas both randomization and subject-pair matchings offer quickly diminishing reductions. Making groups similar before treatment allows for statistical power beyond what can normally be hoped for with small samples.

We propose that optimization protects against experimental biases at least as well as randomization and that the advantage of optimized groups over randomized groups is substantial. We believe that optimization of experimental group composition, implementable on commonplace software such as Microsoft Excel and on commercial mathematical optimization software, is a practical and desirable alternative to randomization that can improve experimental power in numerous fields, such as cancer research, neurobiology, immunology, investment analysis, market research, behavioral research, proof-of-concept clinical trials, and others.

16.7 Notes and Sources

This chapter is from [21].

Chapter 17

Identifying Exceptional Responders

Many of life's failures are people who did not realize how close they were to success when they gave up.

– Thomas A. Edison

Contents

- 17.1. Identifying interpretable optimal subgroups
- 17.2. Computational experiments with synthetic data sets
- 17.3. Computational experiments with real-world datasets
- 17.4. Concluding Remarks
- 17.5. Notes and Sources

Researchers in the medical and social sciences invest substantial resources to implement randomized controlled trials, the gold standard in statistical analysis of response to a treatment. Whether the response measured is biological, economic, social, or otherwise, the hope of randomized trials is to confirm the effectiveness (or harm) of an experimental intervention. When a trial fails to yield a significant result, the investigator may abandon the study of the intervention entirely despite the initial investment made. In this chapter, we present a method that uses optimization to identify interpretable subgroups for which an exceptionally large positive or negative response was found.

Such a method could provide great value in the pharmaceutical industry. For instance, in 2010, the expenditure of global pharmaceutical companies on clinical trials for investigational drugs was \$32.5 billion, due to multiple phases of clinical trials necessary for a drug approval process that typically take years to complete ([17, 262]). It is estimated that, from 2003 to 2011, 60% of Phase III clinical trials for investigational drug indications led to submission of a New Drug Application or Biologic License Application to the U.S. Food and Drug Administration, of which 83% proceeded to approval [130]. Our proposed method could suggest promising subpopulations in which to conduct (or avoid) further testing of the treatment. In this way, there is the potential to increase the value of research and development dollars and realize large-scale economic benefits throughout the healthcare industry. Investigators could also use our method to revisit long-terminated clinical trials and search for opportunities to revive the testing of failed drugs in promising subgroups.

This potential to enhance the value of clinical trials through subgroup identification may exist even for trials that were initially successful. Subgroup identification could point investigators to the prevalence of adverse events arising from the use of new or existing drugs in subpopulations. For approved drugs whose patents are expiring, our method may increase the financial benefit of the company's initial research and development investment by suggesting opportunities for re-marketing the drug to different segments of the population or for different medicinal purposes.

We present a mixed-integer optimization (MIO) approach to identify a subgroup for which the average treatment effect was exceptionally strong or exceptionally weak and which can be defined by a small pre-specified number of covariates. When a randomized clinical trial is unable to reject the null hypothesis of no effect, our method may identify a subset for which the treatment was effective, or provide evidence that there is no such subgroup. Even when the randomized clinical trial is conclusive in confirming the effectiveness or harm of a treatment, our method can identify subgroups for which the treatment was particularly effective or particularly ineffective.

17.1 Identifying interpretable optimal subgroups

Let us consider a randomized controlled trial in which subjects, indexed by $i \in [n]$, have received treatment assignments T_i to one of two treatment conditions: treatment ($T_i = 1$) or control ($T_i = 0$). We define the sets $\mathcal{T}_t := \{i \mid T_i = t\}$, $t = 0, 1$. For each subject, we observe a covariate vector $\mathbf{x}_i \in \mathbb{R}^S$, where S is the number of observed covariates. We also observe treatment responses v_i , $i \in [n]$.

Our objective is to find the subset $\mathcal{I}^* \subseteq [n]$ with maximum (or minimum) average treatment value (ATE), where

$$\text{ATE}(\mathcal{I}^*) := \frac{1}{|\mathcal{T}_1 \cap \mathcal{I}^*|} \sum_{i \in \mathcal{T}_1 \cap \mathcal{I}^*} v_i - \frac{1}{|\mathcal{T}_0 \cap \mathcal{I}^*|} \sum_{i \in \mathcal{T}_0 \cap \mathcal{I}^*} v_i.$$

For the remainder of the chapter, we focus on the maximization problem and we assume that larger values of the response v_i are preferable.

To make the definition of subset \mathcal{I}^* interpretable, for each covariate $s \in [S]$, we define K_s hyperplanes parallel to the coordinate axes that could be chosen as boundaries for the box containing the subset. Each hyperplane is defined by a value γ_{sk} , $s \in [S]$, $k \in [K_s]$, such that

$$\min_i x_{is} - \epsilon = \gamma_{s1} < \gamma_{s2} < \dots < \gamma_{sK_s} = \max_i x_{is} + \epsilon,$$

where $\epsilon > 0$ is a small perturbation term. We limit the number of covariate dimensions along which we can restrict the subset to some number $S^0 \leq S$. We also define the quantities \underline{N} and \overline{N} , which constrain the cardinality of \mathcal{I}^* , such that $\underline{N} \leq |\mathcal{T}_t \cap \mathcal{I}^*| \leq \overline{N}$, $t = 0, 1$.

Mixed-integer optimization

In modeling the problem as an optimization problem, the key decisions are to define the boundaries of the subset. Let $\mathbf{L} := \{L_{sk} \mid s \in [S], k \in [K_s]\}$ be a set of binary decision variables that take value 1, if γ_{sk} is chosen as the lower bound for dimension s , and 0, otherwise. Similarly, let $\mathbf{U} := \{U_{sk} \mid s \in [S], k \in [K_s]\}$ be a set of binary decision variables that take value 1, if γ_{sk} is chosen as the upper bound for dimension s , and 0, otherwise. Taken together, these binary decision variables uniquely define a box in the space \mathbb{R}^S , which will provide an interpretable subset.

We define auxiliary binary decision variables $z_i := \mathbb{I}\{i \in \mathcal{I}^*\}$, $i \in [n]$. To enforce the limit on splitting dimensions, we define auxiliary binary indicator variables q_s , which indicate whether or not covariate dimension s is used to restrict the subset \mathcal{I}^* . Both vectors \mathbf{z} and \mathbf{q} are fully determined by the primary decision vectors \mathbf{L} and \mathbf{U} , according to constraints (17.1b)–(17.1d) and (17.1g)–(17.1i), respectively.

We next describe the following fractional mixed-integer optimization (MIO) to identify the subset with highest ATE:

$$\max_{\mathbf{z}, \mathbf{q}, \mathbf{L}, \mathbf{U}} \frac{\sum_{i \in \mathcal{T}_1} v_i z_i}{\sum_{i \in \mathcal{T}_1} z_i} - \frac{\sum_{i \in \mathcal{T}_0} v_i z_i}{\sum_{i \in \mathcal{T}_0} z_i} \quad (17.1a)$$

$$\text{s.t. } z_i + \sum_{s=1}^S \left[\sum_{k: \gamma_{sk} > x_{is}} L_{sk} + \sum_{k: \gamma_{sk} < x_{is}} U_{sk} \right] \geq 1, \quad \forall i \in [n], \quad (17.1b)$$

$$z_i + L_{sk} \leq 1, \quad \forall s \in [S], k \in [K_s], \quad i : x_{is} < \gamma_{sk}, \quad (17.1c)$$

$$z_i + U_{sk} \leq 1, \quad \forall s \in [S], k \in [K_s], \quad i : x_{is} > \gamma_{sk}, \quad (17.1d)$$

$$\sum_{k=1}^{K_s} L_{sk} = 1, \quad \forall s \in [S], \quad (17.1e)$$

$$\sum_{k=1}^{K_s} U_{sk} = 1, \quad \forall s \in [S], \quad (17.1f)$$

$$q_s + L_{s1} \geq 1, \quad \forall s \in [S], \quad (17.1g)$$

$$q_s + U_{sK_s} \geq 1, \quad \forall s \in [S], \quad (17.1h)$$

$$q_s + L_{s1} + U_{sK_s} \leq 2, \quad \forall s \in [S], \quad (17.1i)$$

$$\sum_{s=1}^S q_s \leq S_0, \quad (17.1j)$$

$$\underline{N} \leq \sum_{i \in \mathcal{T}_t} z_i \leq \bar{N}, \quad \forall t = 0, 1, \quad (17.1k)$$

$$\mathbf{z}, \mathbf{q}, \mathbf{L}, \mathbf{U} \in \{0, 1\}.$$

The constraints in formulation (17.1) deserve further discussion. First, for any given subject i , if the following condition is met,

$$\sum_{s=1}^S \left[\sum_{k: \gamma_{sk} > x_{is}} L_{sk} + \sum_{k: \gamma_{sk} < x_{is}} U_{sk} \right] = 0, \quad (17.2)$$

then z_i must be equal to 1, by constraints (17.1b). We observe that condition (17.2) is met if and only if, for all covariates $s \in [S]$, both of the following statements are true:

1. By our choice of \mathbf{L} , we do *not* select any lower bound γ_{sk} for which $\gamma_{sk} > x_{is}$; and,
2. By our choice of \mathbf{U} , we do *not* select any upper bound γ_{sk} for which $\gamma_{sk} < x_{is}$,

where x_{is} is the value of covariate s for subject i . Taken together, these statements imply that subject i must be in the box defined by the choices of \mathbf{L} and \mathbf{U} . Therefore, by construction, the auxiliary variable z_i should be equal to 1 in this case.

Conversely, constraints (17.1c) and (17.1d) ensure that z_i must be equal to 0 whenever the choices of \mathbf{L} and \mathbf{U} imply that subject i is outside the box. Specifically, if we select γ_{sk} as a lower bound on dimension s by taking $L_{sk} = 1$, then by constraints (17.1c) we have $z_i = 0$ for all subjects i for which $x_{is} < \gamma_{sk}$. If we select γ_{sk} as an upper bound by taking $U_{sk} = 1$, then by constraints (17.1d) we have $z_i = 0$ for all subjects i for which $x_{is} > \gamma_{sk}$.

Constraints (17.1e) and (17.1f) indicate that only one lower and upper bound, respectively, can be chosen for each covariate dimension. Constraints (17.1g), (17.1h), and (17.1i) encode the desired relationships $q_s = 1 - \mathbb{I}\{L_{s1} = 1 \text{ and } U_{sK_s} = 1\}$, $s \in [S]$, so that q_s indicates whether or not dimension s is used to restrict the space of \mathcal{I}^* . This relationship allows us to require that at most S_0 covariate dimensions are used to restrict the subset by adding constraint (17.1j). Finally, constraints (17.1k) ensure that the cardinality of \mathcal{I}^* conforms to the specified limits; note that these constraints also ensure that there will be no dimension for which the lower bound chosen exceeds the upper bound.

Tractable transformation of fractional objective

Because formulation (17.1) has a fractional objective function (17.1a), the problem cannot be solved using off-the-shelf commercial solvers. We can transform the objective from fractional to non-fractional by considering the expressions:

$$\Theta_t := \frac{1}{\sum_{i \in \mathcal{T}_t} z_i}, \quad t = 0, 1. \quad (17.3)$$

By construction, Θ_t , $t = 0, 1$, are discrete variables that take values in the set $\left\{\frac{1}{N}, \frac{1}{N+1}, \dots, \frac{1}{\bar{N}}\right\}$. Therefore, we can represent each discrete variable by the binary expansion

$$\Theta_t = \sum_{j=\underline{N}}^{\bar{N}} \frac{1}{j} \theta_j^{(t)}, \quad t = 0, 1$$

where $\theta_j^{(t)}$, $j = \underline{N}, \dots, \bar{N}$, $t = 0, 1$ are binary variables with $\sum_{j=\underline{N}}^{\bar{N}} \theta_j^{(t)} = 1$, $t = 0, 1$. We can now rewrite the fractional objective function (17.1a) as a non-fractional, nonlinear expression:

$$\Theta_1 \sum_{i \in \mathcal{T}_1} v_i z_i - \Theta_0 \sum_{i \in \mathcal{T}_0} v_i z_i = \sum_{i \in \mathcal{T}_1} \sum_{j=\underline{N}}^{\bar{N}} \frac{1}{j} v_i z_i \theta_j^{(1)} - \sum_{i \in \mathcal{T}_0} \sum_{j=\underline{N}}^{\bar{N}} \frac{1}{j} v_i z_i \theta_j^{(0)}.$$

To make the objective linear, we introduce additional binary variables ζ_{ij} , $i \in [n]$, $j = \underline{N}, \dots, \overline{N}$. To model the desired relationship $\zeta_{ij} = z_i \theta_j^{(T_i)}$, which is the product of two binary variables, we add three sets of constraints (17.4b)–(17.4d): We now have the linear objective:

$$\sum_{i \in \mathcal{T}_1} \sum_{j=\underline{N}}^{\overline{N}} \frac{1}{j} v_i \zeta_{ij} - \sum_{i \in \mathcal{T}_0} \sum_{j=\underline{N}}^{\overline{N}} \frac{1}{j} v_i \zeta_{ij}.$$

Finally, to enforce the stated relationship (17.3), we add the constraints:

$$\Theta_t \sum_{i \in \mathcal{T}_t} z_i = \sum_{i \in \mathcal{T}_t} \sum_{j=\underline{N}}^{\overline{N}} \frac{1}{j} \zeta_{ij} = 1, \quad \forall t = 0, 1.$$

Taking together all of these substitutions and constraints, we obtain the following mixed-binary linear optimization formulation, which is equivalent to formulation (17.1) and can be solved using commercial optimization solvers:

$$\max_{\mathbf{z}, \mathbf{q}, \mathbf{L}, \mathbf{U}, \boldsymbol{\zeta}, \boldsymbol{\theta}} \sum_{i \in \mathcal{T}_1} \sum_{j=\underline{N}}^{\overline{N}} \frac{1}{j} v_i \zeta_{ij} - \sum_{i \in \mathcal{T}_0} \sum_{j=\underline{N}}^{\overline{N}} \frac{1}{j} v_i \zeta_{ij} \quad (17.4a)$$

$$\text{s.t. } z_i + \sum_{s=1}^S \left[\sum_{k: \gamma_{sk} > x_{is}} L_{sk} + \sum_{k: \gamma_{sk} < x_{is}} U_{sk} \right] \geq 1, \quad \forall i \in [n],$$

$$z_i + L_{s1} \leq 1, \quad \forall s \in [S], k \in [K_s], \quad i : x_{is} < \gamma_{s1},$$

$$z_i + U_{sK_s} \leq 1, \quad s \in [S], k \in [K_s], \quad i : x_{is} > \gamma_{sK_s},$$

$$\sum_{k=1}^{K_s} L_{sk} = 1, \quad \forall s \in [S],$$

$$\sum_{k=1}^{K_s} U_{sk} = 1, \quad \forall s \in [S],$$

$$q_s + L_{s1} \geq 1, \quad \forall s \in [S],$$

$$q_s + U_{sK_s} \geq 1, \quad \forall s \in [S],$$

$$q_s + L_{s1} + U_{sK_s} \leq 2, \quad \forall s \in [S],$$

$$\sum_{s=1}^S q_s \leq S_0,$$

$$\underline{N} \leq \sum_{i \in \mathcal{T}_t} z_i \leq \overline{N}, \quad \forall t = 0, 1,$$

$$\zeta_{ij} \leq \theta_j^{(T_i)}, \quad \forall i \in [n], j = \underline{N}, \dots, \overline{N}, \quad (17.4b)$$

$$\zeta_{ij} \leq z_i, \forall i \in [n], j = \underline{N}, \dots, \overline{N}, \quad (17.4c)$$

$$\zeta_{ij} \geq \theta_j^{(T_i)} + z_i - 1, \forall i \in [n], j = \underline{N}, \dots, \overline{N}, \quad (17.4d)$$

$$\sum_{i \in \mathcal{T}_t} \sum_{j=\underline{N}}^{\overline{N}} \frac{1}{j} \zeta_{ij} = 1, \forall t = 0, 1,$$

$$\sum_{j=\underline{N}}^{\overline{N}} \theta_j^{(t)} = 1, \forall t = 0, 1,$$

$$0 \leq \zeta_{ij} \leq 1, \forall i \in [n], j = \underline{N}, \dots, \overline{N},$$

$$\mathbf{z}, \mathbf{q}, \mathbf{L}, \mathbf{U}, \boldsymbol{\theta} \in \{0, 1\}.$$

Note that ζ can be included as continuous variables on $[0,1]$, which improves the computational performance as the number of binary variables is linear in n . Thus, the formulation includes $\mathcal{O}(n^2)$ continuous variables, with $\mathcal{O}(n)$ binary variables, assuming $\sum_{s=1}^S K_s$ is small relative to n .

Statistical significance of optimal subset

Assuming γ is defined in a sensible manner, solving formulation (17.4) using an off-the-shelf commercial optimization solver, such as Gurobi [124], will yield an optimal or near-optimal feasible subset in all practical instances. Yet, it is unreasonable to expect that every randomized trial has a latent subset in which a significant positive or negative effect is observed.

We propose the use of statistical hypothesis testing to determine whether the average treatment effect within the optimal subgroup is statistically significant. We adopt the null hypothesis that there is no difference in treatment response between the treatment groups.¹ We introduce some robustness to outliers by considering the trimmed mean, i.e., the average treatment effect in which the largest 10% and the smallest 10% of response values are discarded from the trial sample. In testing, we found that this trimmed ATE yielded more stable and trustworthy determinations of positive effect size than the standard ATE. If the optimal subgroup has a statistically significant trimmed ATE, we consider the subset to be viable and we recommend additional testing of the treatment in subjects who match the subgroup's covariate profile. Otherwise, we discard the optimal solution and determine that there is no need for further study of the treatment in this population. For the computational experiments in Section 17.2, we use a non-parametric hypothesis testing approach based

¹In settings where the overall study sample had a statistically significant non-zero treatment response, one may want to adopt a different null hypothesis that the treatment effect in the subgroup does not differ from the overall treatment effect in the study population.

on the bootstrap [90], with a significance level of $\alpha = 0.01$, chosen to yield a desired balance between true and false positive rates.

If the sample size n is sufficiently large, one can introduce an additional level of verification by reserving a subset of the data as a test set not to be used when finding the optimal subset. The majority of the data can be used as a training set on which to apply the optimization approach and find an optimal subset. Then, one can identify which subjects from the reserved test set are in the box defined by the optimal solution, and evaluate the out-of-sample ATE and its significance within the test subset.

Finding multiple subsets

Up until now, we solve the problem of finding a single interpretable subset of the data with maximum ATE. Let us assume one wants to find M subsets with the highest ATEs, such that there is limited overlap between the subsets. A greedy approach to this problem is to iteratively solve the MIO formulation from the previous section, while adding a constraint on the overlap with previous optimal subsets. For instance, let $Z_1 := \{i \mid z_i^* = 1\}$ be the set of data points in the optimal subset found in the first iteration of the optimization. One can then seek a second subset Z_2 with maximal ATE but small impurity between Z_1 and Z_2 , such that:

$$\frac{|Z_1 \cap Z_2|}{|Z_1 \cup Z_2|} \leq \rho,$$

for some pre-defined impurity parameter ρ .

In order to find Z_2 that satisfies this impurity constraint, one can add the following constraint to formulation (17.4):

$$\sum_{i \in Z_1} z_i \leq \rho \cdot \left(|Z_1| + \sum_{i \notin Z_1} z_i \right), \quad (17.5)$$

since $|Z_1 \cup Z_2| = |Z_1| + |Z_2 \setminus Z_1|$. Then re-solving the MIO with the added constraint (17.5) yields the optimal Z_2 . More generally, for any $m = 2, \dots, M$, one can add pairwise impurity constraints:

$$\sum_{i \in Z_\ell} z_i \leq \rho \cdot \left(|Z_\ell| + \sum_{i \notin Z_\ell} z_i \right), \quad \forall \ell = 1, \dots, m-1.$$

When testing for significance with multiple subsets ($M > 1$), one should be careful to account for multiple comparisons using the Holm-Bonferroni procedure or a similar approach [139].

Recursive partitioning heuristic

We present a heuristic inspired by the greedy, recursive partitioning scheme of classification trees [54]. The creation of this heuristic is motivated by

the desire to quickly obtain near-optimal solutions, either for direct use or as warm-start feasible solutions for formulation (17.4).

Like CART, at each branch, the heuristic searches for a one-dimensional covariate split that will greedily maximize an objective function, in this case the ATE for subjects in the subset. In our heuristic, the split is determined by choosing both a lower and upper bound describing the subset along a single covariate dimension. There are several parameters that govern the choice of split at each branch in order to ensure heuristic solutions are feasible for formulation (17.4). First, the number of covariates used to make splits in a given tree must not exceed S_0 . Second, for any given tree, we specify a depth parameter d , which serves as an upper bound on the number of branches that can be made in a given tree. Third, at each step of the recursion $c = 1, \dots, d$, where d is the chosen depth parameter, we gradually decrease lower and upper bounds on the cardinality of the leaf indicating the solution subset. The lower and upper bounds are specified as $\underline{N}^c = \frac{n}{2} \cdot (\frac{2N}{n})^{\frac{1}{d-c+1}}$ and $\overline{N}^c = \frac{n}{2} \cdot (\frac{2\bar{N}}{n})^{\frac{1}{d-c+1}}$, respectively, where n is the full sample size and \underline{N} and \overline{N} are the subset cardinality bounds; on the last step of the recursion, when $c = d$, we have $\underline{N}^c = \underline{N}$ and $\overline{N}^c = \overline{N}$, so that the heuristic solution is feasible for formulation (17.4). We define $\delta_H(d)$ to be the ATE in the best subset found after applying the heuristic with depth parameter d .

Rather than generate a single tree yielding a single heuristic solution, we can specify a set of integer depth parameters \mathcal{D} and use the heuristic to find the solution $\delta_H^* := \max_{d \in \mathcal{D}} \delta_H(d)$. For the computational experiments in Section 17.2, we use $\mathcal{D} = \{1, \dots, 8\}$. To consider an even larger range of solutions, we can specify S^2 different starting pairs (s_1, s_2) , $s_1 = 1, \dots, S$, $s_2 = 1, \dots, S$, such that the first branch of the tree must split on dimension s_1 and the second branch must split on dimension s_2 . If $\delta_H(d, s_1, s_2)$ is the objective value of the best tree found with these starting points, then we have $\delta_H(d) := \max_{(s_1, s_2) \in \{1, \dots, S\}^2} \delta_H(d, s_1, s_2)$.

At the end, we take δ_H^* and the corresponding subset lower and upper bounds as our best heuristic solution. In all tested instances, the heuristic found a near-optimal or optimal solution within seconds.

17.2 Computational experiments with synthetic data sets

We conducted simulation experiments with data generated according to several different models relating the response vector \mathbf{v} to treatment vector \mathbf{y} and covariate matrix \mathbf{X} . In the base experiment, we had $n = 100$ subjects with four measured covariates $x_i^{(1)}, \dots, x_i^{(4)}$, $i \in [n]$ drawn *i.i.d.* from a continuous uniform distribution over $[0,1]$. Subjects were randomly assigned to one of two treatment conditions, $y_i = 1$, indicating assignment

to the treatment group, or $y_i = 0$, indicating assignment to the control group, with an equal number assigned to each group. For each experiment, we assume a linear data model

$$v_i = 2 + \delta_0 \cdot y_i \cdot \mathbb{I}\{x_i^{(1)} \leq 0.5\} \cdot \mathbb{I}\{x_i^{(2)} \leq 0.5\} + \varepsilon_i, \quad (17.6)$$

where δ_0 is the ground truth treatment effect and ε_i is a noise term. In the base case, we assume $\delta_0 = 2$ and ε_i drawn *i.i.d* standard normal. To evaluate the false positive rate of our method, we also considered a modification in which $\delta_0 = 0$.

For 250 unique, random samples, we solved formulation (17.4) with $S_0 = 2$, $\underline{N} = [0.1 \cdot n]$, $\overline{N} = [0.3 \cdot n]$, and γ specified from 0 to 1 by increments of 0.1 for all dimensions. The computations were implemented using Julia programming language [45, 183] and the integer optimization solver Gurobi 6.5 [124]. For each sample, we applied a bootstrapped hypothesis test used by [21] with significance level $\alpha = 0.01$ to determine whether the subgroup had a statistically significant treatment response.

In the modified case with $\delta_0 = 0$, the algorithm erroneously identified a statistically significant subset 12.4% of the time, which we consider to be the false positive rate. In the base case with $\delta_0 = 2$, the method identified a statistically significant subset in 76.8% of simulations. However, due to noise in the response model (17.6), the subset identified did not always precisely match the known underlying best subset $\mathcal{I}_0^* := \{i \mid x_i^{(1)} \leq 0.5, x_i^{(2)} \leq 0.5\}$ (Figure 17.1). In order to evaluate the true positive rate of our method in the base case, we compared all significant found subsets to the known best subset. The confusion matrix showing the number of subjects within each subset averaged across all simulations is shown in Table 17.1, for subsets that were found to be statistically significant. According to Table 17.1, the average accuracy, or percent of subjects for which the classification of the found subset matched the best known classification, was 88.5%; the accuracy should be compared with the baseline prevalence of 74.9% of subjects not in \mathcal{I}_0^* .

Table 17.1: Average percent of subjects in found subset versus best known subset \mathcal{I}_0^* over 250 simulations in base case, when found subset was significant.

	$i \in \mathcal{I}_0^*$	$i \notin \mathcal{I}_0^*$
In found subset ($z_i = 1$)	18.5%	4.9%
Not in found subset ($z_i = 0$)	6.6%	70.0%

One way to evaluate the effectiveness of our method is to examine subsets for which the ATE was found to be statistically significant *and* at least 50% of subjects in the found subset ($z_i = 1$) were also in the known best subset ($i \in \mathcal{I}_0^*$), i.e., the positive predictive value (PPV) was

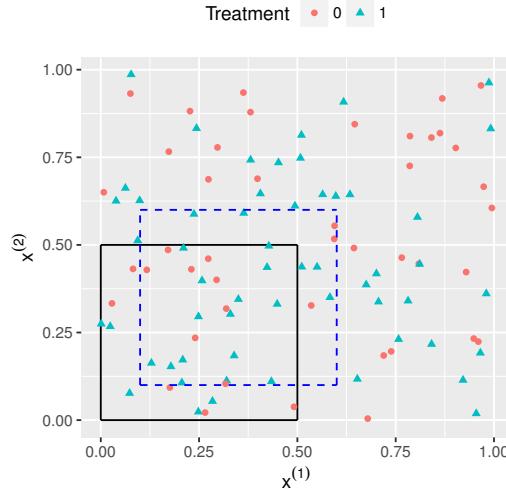


Figure 17.1: An example of a data sample projected onto the $(x^{(1)}, x^{(2)})$ space. Shape indicates treatment group. Black solid line delineates boundary of known best subset \mathcal{I}_0^* . Blue dashed line delineates boundary of found optimal subset.

greater than 50%. We use the term threshold-based true positive rate (TPR) to refer to the percent of found subsets with a significant positive ATE and a PPV greater than 50%. The threshold-based TPR in the base case was 68.0%. Alternatively, if we simply average the PPV among subsets determined to have statistically significant positive ATE, we derive the PPV-weighted TPR, which was 60.7% in the base case. When using the heuristic alone, without mixed-integer optimization, the threshold-based TPR was 66.4% and the PPV-weighted TPR was 59.0%.

We conducted sensitivity analyses with respect to sample size n , covariate dimension S , depth restriction S_0 , effect size δ_0 , and the distribution of the noise term ε_i (Table 17.2). As expected, the TPR (both threshold-based and PPV-weighted) increased with n and δ_0 , and decreased with the variance of ε_i . The TPR was relatively unchanged with respect to S and was very low for $S_0 = 1$, but grew slowly for $S_0 \geq 2$. The false positive rate (FPR) was stable and low in all tested instances, although it grew with respect to S likely due to the increase in the combinatorial space of possible subsets.

We also conducted computational timing experiments to test how tractable the optimization is as n grows. In all tested instances, the heuristic found an optimal or near-optimal warm-start solution within seconds. The time to provable optimality is shown in Table 17.3 for experiments with $S = 10$ and $\sum_{s=1}^S K_s = 110$.

Table 17.2: Sensitivity analyses of true positive rate (TPR) and false positive rate (FPR) of found subsets with varying sample size n , effect size δ_0 , and distributions of noise ε_i .

Par.	Value	Percent subsets significant	TPR based on threshold	TPR weighted by PPV	FPR
n	100*	76.8%	68.0%	60.7%	12.4%
	150	96.8%	90.8%	84.9%	16.0%
	200	98.8%	97.2%	91.7%	14.4%
S	2	72.4%	70.0%	62.4%	5.2%
	3	79.2%	74.8%	65.2%	11.2%
	4*	76.8%	68.0%	60.7%	12.4%
	5	81.2%	71.6%	63.6%	16.0%
	10	86.8%	66.0%	60.2%	25.2%
S_0	1	40.0%	23.2%	20.8%	3.6%
	2*	76.8%	68.0%	60.7%	12.4%
	3	84.0%	73.2%	63.3%	17.6%
	4	88.0%	76.8%	65.6%	19.2%
δ_0	0	—	—	—	12.4%
	1	28.8%	17.2%	15.9%	—
	2*	76.8%	68.0%	60.7%	—
	4	98.8%	96.0%	87.9%	—
ε_i	$N(0, 1)^*$	76.8%	68.0%	60.7%	12.4%
distr.	$U[-\sqrt{3}, \sqrt{3}]$	68.0%	58.8%	52.1%	12.8%

* Base case parameters used: $n = 100$, $S = 4$, $S_0 = 2$, $\delta_0 = 2$, $\varepsilon_i \sim N(0, 1)$ i.i.d.

Table 17.3: Average computational time to achieve provable optimality by sample size n .

n	Time (seconds)
100	33
250	1,070
500	7,280
1,000	14,535

Table 17.4: True positive rate (TPR) with respect to finding each of two known underlying subsets. Note $\mathcal{I}_1^* := \{i \mid x_i^{(1)} \leq 0.5, x_i^{(2)} \leq 0.5\}$ and $\mathcal{I}_2^* := \{i \mid x_i^{(1)} > 0.5, x_i^{(2)} > 0.5\}$.

Subset	δ_0	Percent subsets significant	TPR based on threshold	TPR weighted by PPV
\mathcal{I}_1^*	6	100.0%	97.2%	89.1%
\mathcal{I}_2^*	3	57.2%	54.0%	50.7%

Multiple subsets

We conducted an additional experiment in which we generated data according to the response model:

$$v_i = 2 + 6 \cdot y_i \cdot \mathbb{I}\{x_i^{(1)} \leq 0.5\} \cdot \mathbb{I}\{x_i^{(2)} \leq 0.5\} + 3 \cdot y_i \cdot \mathbb{I}\{x_i^{(1)} > 0.5\} \cdot \mathbb{I}\{x_i^{(2)} > 0.5\} + \varepsilon_i,$$

with ε_i distributed standard normal. We allowed the algorithm to find two subsets ($M = 2$) and measured the significance of each found subset using the Holm procedure with significance level $\alpha = 0.01$. The results of this experiment are shown in Table 17.4. The method virtually always found the exact subset with $\delta_0 = 6$. With respect to the second subset with $\delta_0 = 3$, the method reliably detected the subset in more than 50% of instances.

17.3 Computational experiments with real-world datasets

In this section, we present three examples in which we apply the optimization approach to real-world datasets. In Examples 1 and 2, the method identifies a subset with statistically significant positive average treatment effect despite an overall non-significant negative treatment effect in the study sample; the results are slightly less conclusive in Example 2 than Example 1. In Example 3, the method does not identify a subgroup with a statistically significant positive average treatment effect. We include both examples to demonstrate our method's ability to discover new insights in clinical trial data, while maintaining appropriate discriminatory power when there is no signal to be found. Note that we cannot evaluate a true positive or false positive rate in these experiments because the underlying presence or absence of a subset of exceptional responders is unknown.

Randomized placebo-controlled trial of diethylstilbestrol for late-stage prostate cancer

Diethylstilbestrol is a form of synthetic estrogen that has been used to treat late-stage prostate cancer. [66] discuss data from a randomized trial testing the effect on survival of diethylstilbestrol at three dosage levels (0.2 mg, 1.0 mg, or 5.0 mg) versus placebo, in 502 patients with stage 3 or 4 prostate cancer.² For each patient, the researchers recorded the months of follow-up and the patient's mortality status, along with covariates, including age, weight, medical history, cancer status, and common laboratory measurements.

Taking the group which received 5.0 mg of estrogen and the placebo group, we analyzed 252 subjects using our optimal subset approach with 12 covariates, $S_0 = 3$, $\underline{N} = 25$, and $\bar{N} = 76$. In the study, the 125 subjects who received the 5.0 mg dose of estrogen had an average survival time of 35.0 months from the time of enrollment until death or end of study follow-up, while the 127 subjects in the placebo group had average survival of 35.3 months. The average survival in the treatment group was 0.3 months shorter in the treatment group than in the placebo group, but the effect was non-significant using the bootstrap hypothesis testing approach discussed in Section 17.2 with significance level $\alpha = 0.01$; this hypothesis testing approach and significance level are used to test for effect significance throughout the current section. Applying our approach, we found an optimal box containing 59 subjects (30 in the estrogen group, 29 in the placebo group). Within this subset, average survival was 42.5 months in the treatment group versus 24.3 months in the placebo group, an average treatment effect of 18.2 additional months of survival. The effect was statistically significant with $p = 0.001$. The subset was defined by patients who have stage 4 cancer, no history of cardiovascular disease, and diastolic blood pressure of 70 mmHg or above at time of measurement. The results suggest that further testing of the 5.0 mg diethylstilbestrol treatment is warranted in subjects meeting these specific criteria.

Randomized controlled trial of Periodontal Therapy to Prevent Pre-term Birth

In the Obstetrics and Periodontal Therapy (OPT) study, researchers studied whether mechanical periodontal therapy administered to pregnant women between 13 and 17 weeks of gestation affected birth outcomes, including gestational age and the presence of congenital anomalies [138].³ The 413 subjects in the treatment group underwent scaling and root planing before 21 weeks, underwent monthly tooth polishing, and received oral

²The [66] data are available at <http://biostat.mc.vanderbilt.edu/wiki/Main/DataSets>.

³The Hedges and Michalowicz data are available at <http://conservancy.umn.edu/handle/11299/160551>.

hygiene instruction. The 410 subjects underwent scaling and root planing after delivery. The study found that, while periodontal treatment improved periodontal outcomes and did not cause harm, the treatment did not significantly alter birth outcomes.

We applied our optimal subset approach to a randomly selected training set of 500 subjects with 11 covariates, including age, body mass index, presence of hypertension and diabetes, use of alcohol and tobacco, number of previous pregnancies, and measures of periodontal disease. The parameters used were $S_0 = 3$, $\underline{N} = 50$, and $\bar{N} = 150$. We considered four different birth outcomes (early birth before 37 weeks, presence of congenital anomalies, and Apgar score at 1 and 5 minutes) and found a significant subset only for the presence of congenital anomalies. Our randomly selected training set included 247 subjects who received the treatment, among whom there were 9 congenital anomalies (3.6%), and 253 subjects from the control group, among whom there were 6 congenital anomalies (2.4%); the average treatment effect was +1.2%, a net increase in anomalies for the treatment group, which was a statistically significant difference at $p < 0.001$. Our method identified a subset including subjects of age 22 to 30, with body mass index ≥ 26 , and fewer than 20 teeth qualifying as having periodontal disease. The subset included 58 subjects who received the treatment, among whom there was 1 anomaly (1.7%) and 51 subjects from the control group, among whom there were 5 anomalies (9.8%). In this subset, the prevalence of congenital anomalies was lower in the treatment group by 8.1%, an average treatment effect that was statistically significant at $p < 0.001$.

We also evaluated outcomes among the remaining 323 subjects not included in the training set. This validation set included 166 subjects from the treatment group with 4 observed anomalies (2.4%), and 157 subjects from the control group with 1 observed anomaly (0.6%). The overall average treatment effect in the validation set was +1.7%, with a higher prevalence of anomalies in the treatment group. Within the validation set, there were 83 subjects who met the criteria to be included in the box identified by our method. In the validation set, the box included 43 treatment group subjects with 0 observed anomalies and 40 control group subjects with 0 observed anomalies. Thus, out of sample, the average treatment effect in the box was zero, which was a statistically significant difference ($p < 0.001$) compared with the 1.7% higher treatment-group prevalence observed for the overall validation set.

While the reduction in the prevalence of anomalies for training set subjects in the box does not fully hold in the validation set, we conclude there is moderate potential that mechanical periodontal therapy can reduce congenital abnormalities among women ages 22 to 30, with BMI ≥ 26 , and fewer than 20 teeth with periodontal disease.

Randomized placebo-controlled trial of D-penicillamine for primary biliary cirrhosis of the liver

Primary biliary cirrhosis (PBC) of the liver is a rare chronic disease that leads to death. Between 1974 and 1984, the Mayo Clinic conducted a placebo controlled trial of the drug D-penicillamine of 312 PBC patients, as described in [98].⁴ For each patient, the researchers recorded the number of days between study registration and the earlier of death, liver transplantation, or the end of the study in July 1986. There were 16 covariates measured at the time of registration, including age, sex, disease stage, presence of associated conditions, and various hematological laboratory measurements, such as serum bilirubin, serum cholesterol, albumin, and platelet counts. Analysis of the study found that there was no significant difference in survival time between the treatment and placebo groups. Among 154 subjects in the placebo group, the average survival time from study enrollment was 1996.9 days. Among 158 subjects in the treatment group, the average survival was 18.7 days longer, at 2,015.6 days. This result was not significant under the null hypothesis using the bootstrap hypothesis test with significance level $\alpha = 0.01$.

We applied our optimal subset approach to determine if there was an interpretable subset of the population for which the drug may have had a significant effect on survival. Because of some missing data, we used 14 of the 16 covariates. We sought an interpretable subset with $S_0 = 3$, $\underline{N} = 20$, and $\bar{N} = 60$. The optimal interpretable subset included 65 subjects who met all three of the following conditions: had not exhibited spider angiomas, had serum bilirubin between 0.75 and 1.5 mg/dL, and had a prothrombin time of no more than 11.1 seconds. In the optimal box, the average survival time among 33 subjects in the treatment group was 2,910.6 days, which was 854.7 days longer than the average survival of 2,055.9 days among 32 subjects in the placebo group. We considered the null hypothesis that the treatment effect in the subset did not differ from that in the overall sample, which we observed to be 18.8 days of added survival. The p -value was 0.03, which was not significant at $\alpha = 0.01$. Therefore, we determined that the subset may have been a false positive and did not warrant further investigation as a possible group of exceptional responders.

17.4 Concluding Remarks

In this chapter, we show that the problem of identifying one or more interpretable subsets of a trial population with best (or worst) average treatment response can be modeled and efficiently solved using mixed-integer linear optimization. We use variable substitution and binary expansion to transform the fractional objective function into a linear

⁴The [98] data are available online at <https://www.umass.edu/statdata/statdata/data/>.

function that is tractable in practical instances. We present an approach for determining whether the found subset is statistically significant. We also introduce a tree-based heuristic that finds near-optimal solutions quickly that serves as a warm start to the mixed-integer optimization solver. In simulated and real-world scenarios we demonstrate that the method finds subgroups worthy of further investigation, while minimizing the rate of false positive subsets.

17.5 Notes and Sources

This chapter is from [23]. The classical statistical approach to identifying subgroups with distinct treatment effects involves using a Cox proportional hazards model with treatment-covariate interaction terms [238]. Citing the disadvantage of needing to specify relevant interactions, [157] improve upon this model by incorporating a bump-hunting procedure based on [104], which they also contrast with the greedier approach of regression trees [54]. [102] introduce several variations of a “virtual twins” method, based on counterfactual modeling and the application of machine learning approaches, including logistic regression, random forest, and classification and regression trees (CART) [58, 54]. This virtual twins method is shown to be effective in simulation studies, with positive predictive value ranging from 45-60%. Others [247, 127] have presented CART-inspired recursive partitioning approaches to solve the subgroup identification or partioining problem. [247] show their method is effective in identifying subgroups with significant positive or negative treatment effect from observational data. The [127] approach has been used in practice to identify subgroups of patients with type 2 diabetes mellitus for which short-acting insulin may provide a benefit. These bump-hunting and tree-based approaches can identify satisfiyingly interpretable subgroups, but they entail greedily solving a sequence of optimization problems.

Chapter 18

Interpretable Clustering

Live with the data before you plunge into modelling.

– Leo Breiman



Chapter 19

The Bootstrap

If there was ever an idea in statistics which evokes the reaction, “Why the hell didn’t I think of that,” it has to be the bootstrap.

– James R. Thompson



Part V

Matrix Methods

Chapter 20

Sparse Principal Component Analysis

—



Chapter 21

Factor Analysis

—



Chapter 22

Sparse Inverse Covariance Estimation

—



Chapter 23

Matrix Completion

—



Chapter 24

Learning from Tensors

—



Part VI

**Machine Learning in
Action**

Chapter 25

Diagnostics for Children After Head Trauma

In God we trust. All others must bring data.

– W. Edwards Deming

Contents

- 25.1. Data and Methods
- 25.2. Results
- 25.3. Discussion
- 25.4. Notes and Sources

Between 2006 and 2010, an estimated 750,000 emergency department visits pertained to pediatric head trauma [185, 186]. Most children with head injury have apparently minor trauma [186, 95]. 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 [101], but it is costly, may require sedation, and exposes patients to ionizing radiation which may increase the risk of malignancies later in life [61, 60, 191]. 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 [164]. Avoiding needless CTs in such patients is highly desirable [164]. 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 [164]. The PECARN prediction tools are widely used [278, 236] and have been independently validated multiple times [182, 148].

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 8 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 [164].

25.1 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 [164]. 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 [164].

We followed the original analysis and stratified the cohort into 10,718 (25%) “younger” (younger than 2 years and predominantly non-verbal) 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 [164]. 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 [164].

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 [164].

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 25.1). 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 8 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 compared estimates from OCT and from

Table 25.1: 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
severity [o4]	sleepy [p2]	paralyzed [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]	

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 25.1, 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* [178]. We imputed missing values using a novel method developed in [43] 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 [43].

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 [164]. All results from sensitivity analyses were congruent with those from the main analysis and are not shown.

25.2 Results

With the PECARN tool (Figure 25.1), 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 corresponding OCT in Figure 25.2 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 25.3). The corresponding OCT in Figure 25.4 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

Figure 25.1: PECARN rules for ciTBI in children younger than 2 years.

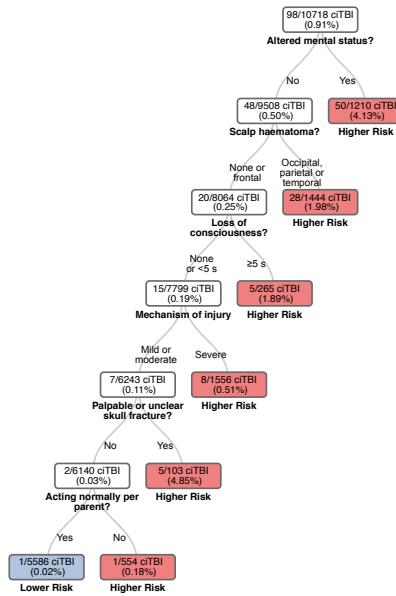


Figure 25.2: OCT for ciTBI in children younger than 2 years.

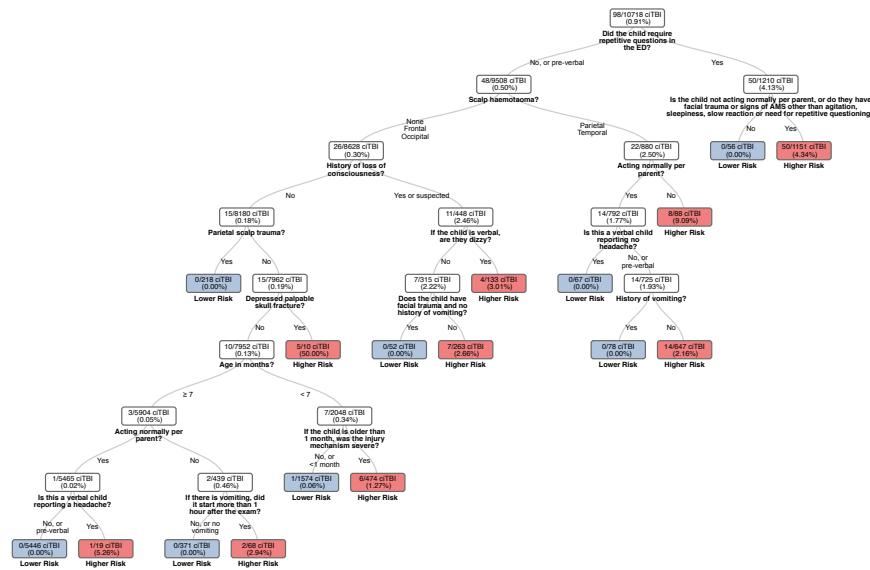
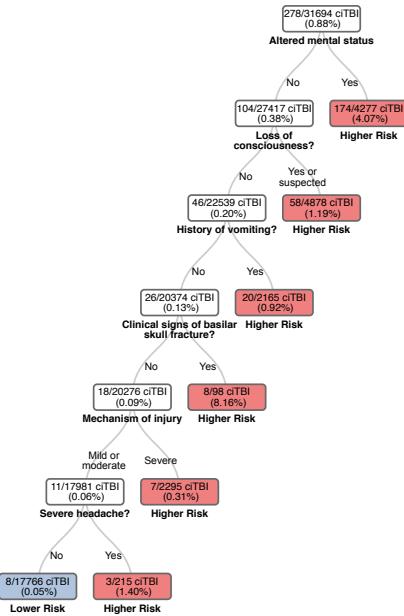


Figure 25.3: PECARN rules for ciTBI in children at least 2 years old.

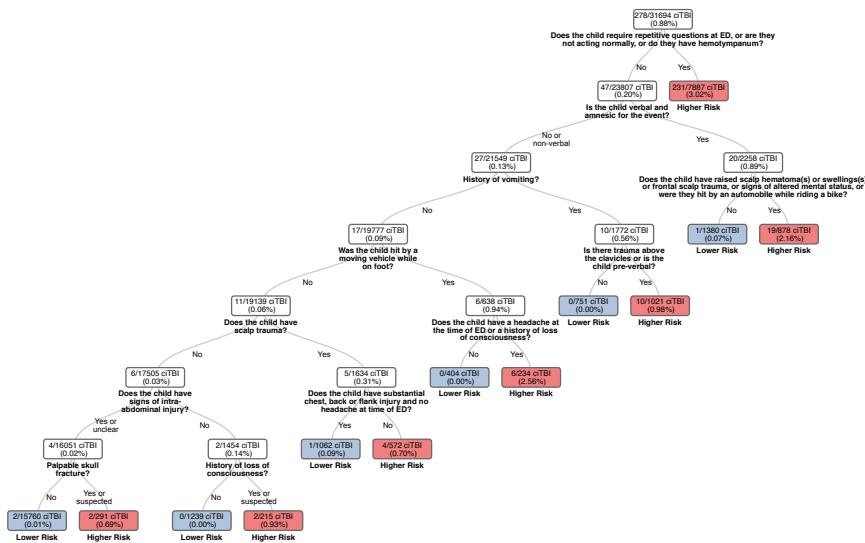


history of vomiting would be classified as having low risk of ciTBI if there is no supraclavicular evidence of trauma (e.g., lacerations, bruises).

The counts of children predicted to be at higher versus low risk of ciTBI with OCT and the PECARN tools versus the true ciTBI status are shown in Tables 25.2 and 25.3 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 25.4). However, OCT outperformed the PECARN rule in correctly identifying more children at low risk of ciTBI. As shown in Table 25.5, 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

Figure 25.4: OCT for ciTBI in children at least 2 years old.

value, and a more favorable negative likelihood 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 25.4). 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 25.4). 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

Table 25.2: 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 25.3: 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

so; Table 25.6).

Table 25.4: 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) Small (<1 cm) parietal or temporal hematoma, moderate headache within 1 h	Hospitalization + TBI in CT
F	>=2 y, training	PECARN rules	6 y (verbal)	Wheeled transport crash other than bike or motor vehicle (moderate)	15	Large (>3 cm) frontal hematoma, moderate headache within 1 h, facial trauma	Hospitalization + TBI in CT
G	>=2 y, training	PECARN rules	15 y (verbal)	Sports (moderate)	15	Anesthesia for the injury, facial trauma	Hospitalization + TBI in CT
H	>=2 y, training	PECARN rules	11 y (verbal)	Sports (moderate)	15	Amnesia for the injury, loss of consciousness for 1-5 mins, moderate headache	Hospitalization + TBI in CT
I	>=2 y, training	OCT rules	11 y (verbal)	Bike collision or fall from bike (moderate)	15	Medium (1-3 cm) parietal or temporal hematoma	Hospitalization + TBI in CT
J	>=2 y, test	OCT, PECARN rules	26 mo (preverbal)	Fell from an elevation (moderate)	15		Hospitalization + TBI in CT

Table 25.5: Predictive performance of OCT versus PECARN rules among children younger than 2 years.

Metric	OCT	PECARN	Odds Ratio
<i>Training</i>			
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)
<i>Test</i>			
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 25.6: Predictive performance of OCT versus PECARN rules among children at least 2 years old.

Metric	OCT	PECARN	Odds Ratio
<i>Training</i>			
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)
<i>Test</i>			
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)

25.3 Discussion

TBI has been deemed a ‘serious public health concern’ by the Centers of Disease and Prevention (CDC) [95]. 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 [186]. 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 [60, 61, 191].

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 Impor-

tant Clinical Events (CHALICE) [86] and the Canadian Assessment of Tomography for Childhood Head Injury (CATCH) rule [210] 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% [88]. If the odds ratios from our head-to-head comparison between OCT and PECARN (Tables 25.5 and 25.6) transfer to the validation study of Easter et al. [88], 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 [213, 216]. Although the PECARN tool was the most sensitive of the examined clinical prediction rules, which included CHALICE [86], the Atabaki et al. tool [3], a University of California Davis tool [212], and the National Emergency X-Radiography Utilization Study (NEXUS) II [198], 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 [213, 216].

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 [50]. 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 [156].

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 [148, 182]. 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.

25.4 Notes and Sources

The material in this chapter is from [39].

Chapter 26

Prediction of mortality for liver transplant allocation

We don't have enough solid organs for transplantation; not enough kidneys, livers, hearts, lungs. When you get a liver and you have three people who need it, who should get it? We tried to come up with an ethically defensible answer. Because we have to choose.

– Ezekiel Emanuel

Contents

- 26.1. Data and Methods
- 26.2. Results
- 26.3. Discussion
- 26.4. Notes and Sources

The successful clinical application of liver transplantation has generated a discrepancy between supply and demand, and in doing so, has generated a persistent insufficient organ supply that results in thousands of candidate deaths every year while awaiting liver transplantation. Given the scarcity of this resource, one of the most crucial challenges in liver transplantation involves accurately predicting a waitlisted candidate's likelihood of death within the near future, so that the limited supply of donated livers can be prioritized and allocated to maximize the benefit from transplantation.

Since 2002, liver allocation has depended on the Model for End-Stage Liver Disease (MELD) score to rank disease severity and, consequently, priority for receiving a liver transplant [222]. Certain patient populations, however, are at risk of death based upon disease progression that is not captured in their lab-based MELD score calculation. To allow them to contend for liver offers, these candidate populations have been granted "artificial" points (MELD exception points). Although overall the MELD score has allowed for a more objective ranking of candidates awaiting liver transplantation, compared to the pre-MELD era, the process of MELD exception point granting has emerged as a significant weakness in the allocation process, leading to inequitable and undesirable outcomes [1]. In particular, the arbitrary MELD score exception points policy has overly prioritized the subpopulation of liver transplant candidates with hepatocellular carcinoma (HCC) [93]. Indeed, since the adoption of the MELD score, there have been multiple policy revisions to reduce the amount of exception points for HCC candidates to more accurately reflect this population's risk of waitlist removal from death or tumor progression. Notwithstanding these revisions, there remains a higher risk of waitlist death/removal for candidates without exception points, when compared to those candidates with exception points.

In this chapter, we utilize Optimal Classification Trees to generate a more accurate prediction of a liver candidate's three-month waitlist mortality or removal, that would in-return allow for a more appropriate prioritization of candidates awaiting liver transplantation. The following prediction problem was posed: *What is the probability that a patient will either die or become unsuitable for liver transplantation within three months, given his or her individual characteristics?*

26.1 Data and Methods

Data

Waitlist, deceased-donor, transplant, and follow-up information was obtained for the period January 1st, 2002 to September 5th, 2016 from the Organ Procurement and Transplantation Network Standard Transplant Analysis and Research (STAR) dataset.

Table 26.1: The independent variables recorded for each observation. The OPOM model considering patients without exception points was trained using variables 1–25 as independent variables; the model considering non-HCC exception patients was trained using variables 1–26; the model considering HCC exception patients was trained using variables 1–29.

ID	Variable name	ID	Variable name
1	Albumin level	16	Dialysis at previous check-in or not
2	Serum bilirubin	17	Change in bilirubin level since previous check-in
3	Serum creatinine	18	Change in creatinine level since previous check-in
4	INR	19	Change in INR since previous check-in
5	Serum sodium level	20	Change in albumin level since previous check-in
6	Dialysis in the last week or not	21	Change in sodium level since previous check-in
7	Number of years accrued on the waitlist	22	Change in Lab MELD score since previous check-in
8	Age in years	23	Log of the candidate's bilirubin level
9	Lab MELD score provided by SRTR	24	Log of the candidate's creatinine level
10	Albumin level at previous check-in	25	Log of the candidate's INR
11	Serum bilirubin at previous check-in	26	Candidate listing region
12	Serum creatinine at previous check-in	27	AFP
13	INR at previous check-in	28	Number of tumors
14	Serum sodium at previous check-in	29	Sum of size of tumors
15	Lab MELD score at previous check-in		

Prediction Methods

The prediction problem was addressed by training OCT models on historical data. The model aimed to predict the probability of a patient dying or becoming unsuitable for transplant within three months, given their characteristics. Henceforth the model is referred to as Optimized Prediction of Mortality (OPOM).

Observations, Dependent and Independent Variables

An observation corresponded to a patient at the time of a check-in visit, so that observed characteristics were all up-to-date. All such available observations from the data for patients aged more than 12 years were retrieved. For each observation, the dependent variable was set to 1 if the patient died or was removed from the waitlist as unsuitable for transplant within the three-month follow-up period from the observation date, and to 0 otherwise. Observations dated before the implementation of MELD, and observations for which patients received a liver transplant during the follow-up period were not considered. In total, 1,244,300 observations were considered. The independent variables recorded for each observation are described in Table 26.1.

what is AFP?

Model Calibration

OPOM comprises three models that consider different patient populations based on exception status: patients without exception points, non-HCC

Figure 26.1: Top part of OPOM Classification Trees for patients with no exception points.

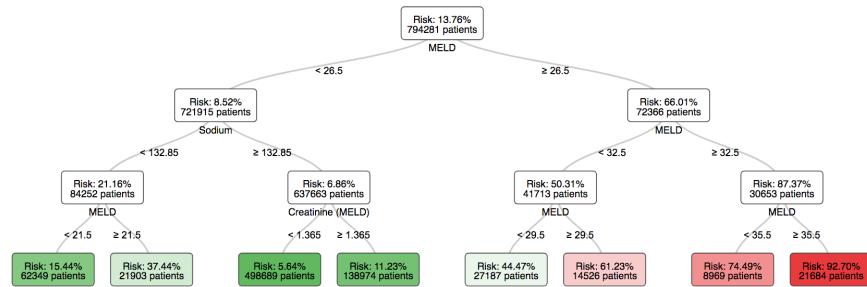
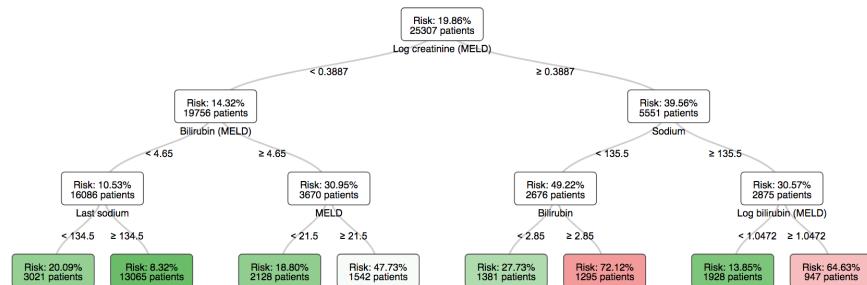


Figure 26.2: Top part of OPOM Classification Trees for patients with non-HCC exception points.



exception patients, and HCC exception patients. To make predictions for a particular patient, OPOM uses the model that matched the patient's exception status.

Observations were randomly split into training, validation, and testing sets. Specifically, 50% of each year's observations were assigned to the training set, 20% to the validation set and 30% to the testing set. OCT models were fit on the training set and then the out-of-sample accuracy value for the validation set was computed. Models with different tree depths (1 to 10) and different numbers of minimum observations in the leaves (1, 5, or 10) were computed, and the models that yielded the highest accuracy for the validation set were selected.

All models are available at <http://www.opom.online>. The top three layers of the selected models by candidate exception status are depicted in Figures 26.1–26.3. Assessment of the independent variables that contributed the most predictive power are demonstrated in Figures 26.4–26.6.

Figure 26.3: Top part of OPOM Classification Trees for patients with HCC exception points.

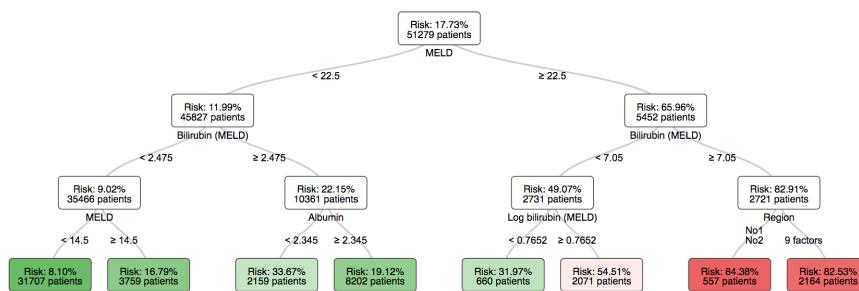


Figure 26.4: Variable importance plots for patients with no exception points.

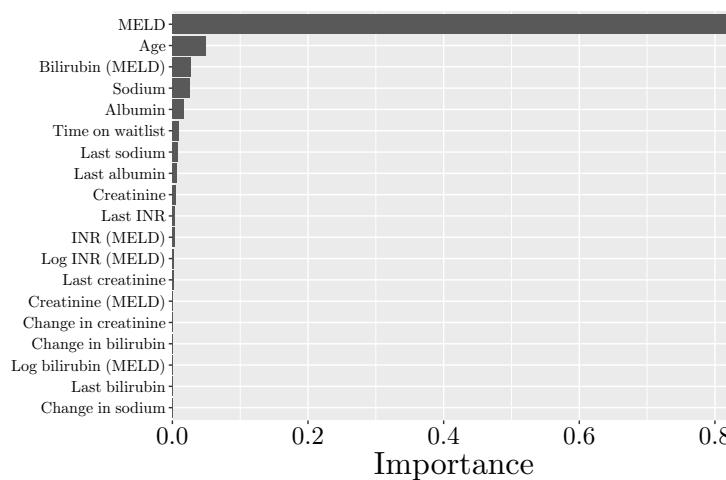


Figure 26.5: Variable importance plots for patients with non-HCC exception points.

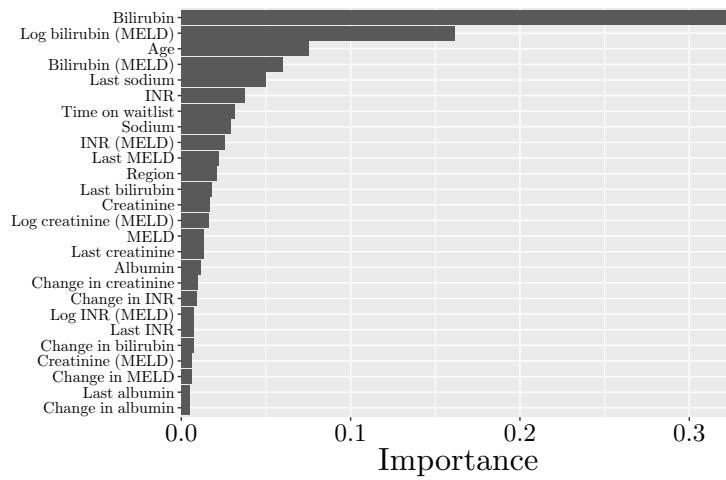
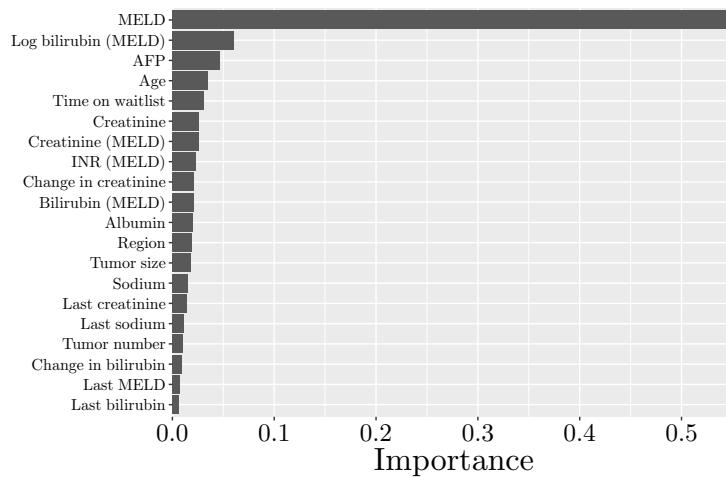


Figure 26.6: Variable importance plots for patients with HCC exception points.



Out-of-sample AUC

To simulate real-world testing procedures, performance was evaluated by measuring out-of-sample Area Under the Curve (AUC) on the testing set, considering the different patient populations based on exception status.

For benchmarking purposes, we also measured on the testing set the AUC of the following MELD variants:

1. Match MELD: MELD score at transplant with consideration of exception points as per the 2014 national allocation policy;
2. MELD-Na: MELD score based on lab values, with no consideration of exception points, but with inclusion of the serum Sodium level.

Allocation Outcomes

The latest version of Liver Simulation Allocation Model (LSAM, version 2014), a program developed by the Scientific Registry of Transplant Recipients¹ that uses historical real-world data from 2007-2011 to simulate the allocation of livers to candidates during that period, was used. LSAM simulates allocations based on Match MELD during the simulation period. To measure the impact of OPOM, the simulation was run after substituting all patients Match MELD scores with their corresponding OPOM scores, which, for consistency, were re-scaled to range between 6-40, instead of 0-100%.

26.2 Results

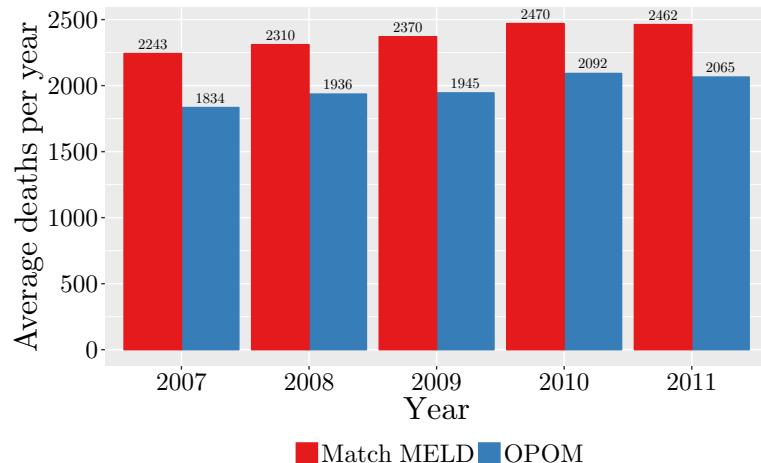
Out-of-sample AUC

The out-of-sample AUC for Match MELD, MELD-Na, and OPOM models is demonstrated in Table 26.2. OPOM considerably outperformed both MELD variants, for all patient exception statuses, when predicting three-month probability of dying or becoming unsuitable for transplant. As a robustness check, the out-of-sample AUC of the models for the subpopulation of patients who had Match MELD scores higher than 25 during the entire observation period was calculated, given that more high-MELD patients receive transplants than low-MELD patients. Table 26.2 reports the out-of-sample AUC of the models, demonstrating that OPOM considerably outperformed all other models for the high-MELD subpopulation.

¹ This study used data from the Scientific Registry of Transplant Recipients (SRTR). The SRTR data system includes data on all donor, wait-listed candidates, and transplant recipients in the US, submitted by the members of the Organ Procurement and Transplantation Network (OPTN). The Health Resources and Services Administration (HRSA), U.S. Department of Health and Human Services provides oversight to the activities of the OPTN and SRTR contractors.

Table 26.2: Out-of-sample AUC by candidate exception status.

Population	Patient type	Match MELD	MELD-Na	OPOM
All Candidates	No Exception	0.835	0.846	0.864
	Non-HCC Exception	0.542	0.795	0.808
	HCC Exception	0.547	0.791	0.805
Match MELD >25 candidates	No Exception	0.725	0.770	0.814
	Non-HCC Exception	0.544	0.803	0.817
	HCC Exception	0.509	0.771	0.789

Figure 26.7: LSAM simulated average deaths by year: Match MELD vs OPOM.

Simulation Results

Figure 26.7 depicts the simulated average number of patient deaths each year by Match MELD and OPOM. Allocation of livers based upon OPOM scores, rather than Match MELD, resulted in 396.6 (16.7%) fewer deaths each year. The demographic profiles of candidates transplanted through OPOM allocation vs. Match MELD is demonstrated in Table 26.3. Notably, a higher number of female candidates received transplants when OPOM allocation was utilized. Further analysis demonstrated that OPOM reduced the number of deaths across all United Network for Organ Sharing (UNOS) Regions when compared to Match MELD. (The range of reductions was 13.6 – 22.5%, and the full results are shown in Figure 26.8)

The simulated average annual number of deaths (waitlist deaths, removed patients' deaths, and post-transplant deaths) by patient status for both models is demonstrated in Table 26.4. Compared to Match MELD, OPOM decreased deaths of waitlisted candidates by 21.8%, decreased

Figure 26.8: LSAM simulated annual percent decrease in deaths by UNOS Region using OPOM liver allocation vs Match MELD.

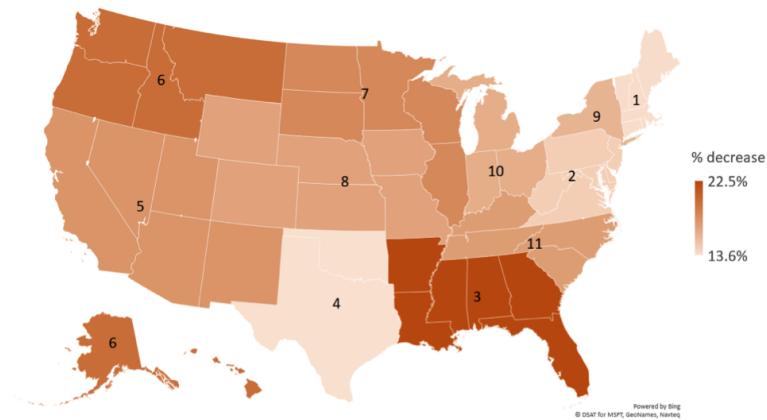


Table 26.3: LSAM simulated average annual deaths and transplants by candidate demographics.

Feature	Value	Match MELD Transplants	OPOM Transplants	Match MELD Deaths	OPOM Deaths
Gender	Male	4003.6	3777.8	1477.6	1212.6
	Female	2154.8	2342.8	893.4	761.8
Race	White	4226.0	4281.0	1673.8	1392.2
	Black	703.6	643.6	216.6	176.6
	Hispanic	857.4	883.4	364.2	310.4
	Asian	298.2	237.8	89.4	72.2
	Other	73.2	74.8	27.0	23.0
Blood type	O	2761.2	2787.8	1148.8	962.8
	A	2187.4	2184	914.2	755.4
	B	848.4	832.6	246.8	207.2
	AB	361.4	316.2	61.2	49.0
Cause of Liver Disease	Acute Hepatic Necrosis	383.0	382.8	103.0	92.0
	Cholestatic Liver Disease	447.6	491.2	163.2	130.4
	Malignant Neoplasms	642.0	394.0	150.2	109.0
	Non-Cholestatic Cirrhosis	3797.4	4075.2	1694.0	1420.6
	Other	888.4	777.4	260.6	222.4
Population averages	Age	50.3	52.6	54.7	54.5
	Cumulative waiting time (days)	174.4	268.4	364.4	385.7
	BMI	27.7	28.0	28.2	28.3

Table 26.4: LSAM simulated average annual deaths and transplants by candidate status and exception status.

Quantity	Subgroup	Match MELD	OPOM
Deaths by patient status	Waitlist Deaths	1207.6	944.0
	Removed Patient Deaths	593.6	473.0
	Post Graft Deaths	569.8	557.4
	All patients	2371.0	1974.4
Deaths by patient exception status	HCC patients	288.6	212.0
	Non-HCC patients	2082.4	1762.4
	All patients	2371.0	1974.4
Removals by patient exception status	HCC patients	260.8	244.0
	Non-HCC patients	2196.0	2007.2
	All patients	2456.8	2251.2
Transplants by patient exception status	HCC patients	1217.2	692.0
	Non-HCC patients	4941.2	5428.6
	All patients	6158.4	6120.6

deaths of candidates removed from the waitlist by 20.3%, and decreased post-transplant deaths by 2.2%. OPOM allocated more livers to non-HCC patients, and less to HCC patients, when compared to Match MELD. However, OPOM, when compared to Match MELD, decreased the number of waitlist deaths and removals for both HCC patients and non-HCC patients. The overall number of transplants performed was simulated to remain stable when OPOM allocation was compared to Match MELD (6120.6 versus 6158.4).

26.3 Discussion

For almost two decades now, MELD has served as the scoring system used to rank liver transplant candidates on the waitlist, albeit with varying levels of success in predicting mortality for different patient populations. Indeed, although a simple method to stratify candidates awaiting liver transplantation, the MELD score is a linear regression method that does not accurately predict mortality for all candidates who can benefit from liver transplantation. In particular, the use of MELD exception points within the current scoring system has represented an arbitrary, yet advantageous, solution for certain sub-populations of candidates, most notably those candidates with HCC. The latter “HCC advantage” has been addressed through first serial downgrades in the amount of MELD exception points granted, and subsequently, more recently, with both a delayed initiation of MELD exception points (6-month delay), as well as a cap on the extent of points an individual can achieve (MELD 34 cap) [134]. These modifications

have been implemented with the hopes of decreasing waitlist mortality and increasing transplant rates in the non-HCC population; however, they have thus far represented insufficient and inexact changes in adequately equalizing access to liver transplants for the non-HCC population.

Optimized Prediction of Mortality (OPOM), a novel system based on OCT, has allowed for the most accurate prediction of three-month mortality rate for all patients on the liver transplant waitlist. OPOM allocation improved the currently-used MELD-based prediction method. In simulations, OPOM averted significantly more waitlist deaths/removals for both HCC and non-HCC candidates, and yet maintained overall transplant rates, therefore allowing for more equitable and efficient allocation of liver grafts for candidates awaiting transplantation. While it is the case that the MELD score and its components (bilirubin, INR, and creatinine) are highly effective predictors of three-month mortality, they are by no means the only relevant predictors. As demonstrated using LSAM, the use of OPOM in place of current Match MELD scores, would save on average at least 397 more lives each year, with every UNOS Region benefiting from this effect. Importantly, the overall number of transplants remains stable with OPOM allocation; albeit with an acceptable, and expected, decrease in HCC transplants to accommodate the increase in transplants of non-exception point candidates. Unlike MELD allocation which relies upon the cumbersome and inexact approach of exception point assignment, OPOM allows for accurate prioritization of all candidates based upon individual characteristics.

The accurate prediction of an individual candidate's risk of waitlist mortality/removal is paramount to ensure equitable access to liver transplantation. Whereas on the one hand MELD-based allocation with inclusion of exception points has over prioritized exception point candidates at the expense of those candidates listed with lab MELD scores, on the other hand simply utilizing a lab-based MELD score for waitlist prioritization would shift the pendulum in the complete opposite direction, resulting in an allocation process that greatly underserves those in need of a liver transplant but with a lab MELD score that does not reflect their severity of disease. OPOM achieves an evidence-based, unbiased and objective middle ground for all waitlisted candidates. Notably, there is a higher number of transplants in the female population with OPOM allocation; perhaps overcoming the systematic bias noted in MELD based allocation for female candidates [87]. The latter has been attributed to the inability of MELD to accurately capture the female candidates degree of renal insufficiency based on serum creatinine levels, resulting in lower MELD scores, and thus lower transplantation rates. OPOM has provided a more complete picture of the individual candidates true waitlist mortality, that in return has allowed for a more accurate prediction of need for liver transplantation.

The 397 waitlist deaths averted with OPOM utilization is significantly more than the number achieved with implementation of MELD-Na. Indeed,

MELD-Na which was approved by UNOS in June 2014 and implemented in January 2016, was predicted through similar LSAM analyses to decrease waitlist deaths by only 52 patients a year [93]. Similarly, the application of a 6-month delay in awarding exception points for HCC candidates was simulated in LSAM to achieve a higher rate of transplants for the non-HCC candidates, at the expense of a lower transplant rate for HCC candidates [134]. The downstream effect on waitlist mortality with this proposed change, compared with the current policy, was approximately 40 more deaths among HCC candidates and 70 fewer deaths among non-HCC candidates, a net reduction of only 30 deaths overall. The latter policy was adopted in October of 2015, and much like the acceptance of MELD-Na, although well intentioned, has overall represented nominal changes in waitlist mortality in simulations when compared to liver allocation through OPOM.

Decision trees have the potential to become an indispensable tool for clinicians with optimized predictions based upon large amounts of data [209]. We utilized OCT to develop an analytical tool that takes all available patient information to predict as to whether the patient will undergo the adverse events of either death or becoming unsuitable for transplant within three months. In contrast to the piecemeal way in which current policy has been constructed, our tool is trained on historical outcomes in a unified fashion utilizing millions of data points. Instead of adding in exceptions and cutoffs ex-post to decrease mortality on the waitlist, machine-learning analytical tools tackle the problem directly by building these different criteria into the model itself. The out-of-sample AUC and accuracy illustrated that OPOM performs well not only on patients without exceptions, but also on patients with HCC exceptions and non-HCC exceptions.

Limitations of this study include the fact that OCT models retain “discontinuities,” similar to other classification-tree-based methods of machine learning. However, this effect is dampened with OCT by the large number of branches and leaves allowing for a more robust analysis. The use of readily available, reproducible, and objective data that accurately predict liver-related mortality is essential. Although OPOM utilizes a large number of variables in generation of the OCT models, no additional data collection would be required by the transplant practitioner, as the OCT models were generated based upon available data within the STAR files. Furthermore, OCT is a versatile and dynamic tool that can allow for additional variables to be included/excluded with ease should additional priorities in liver allocation require that the OCT be modified. It should be noted that LSAM analysis is limited in that it only allows for an accurate assessment of waitlist deaths, as waitlist removals include not only candidates with deterioration in their condition, but also those removed due to improvement in their condition. Granularity in the varying types of exceptions within LSAM would also allow for a more accurate assessment

of the differing classes of exception point candidates, instead of a simple HCC versus no-HCC candidate comparison. Although additional analysis with consideration of a shorter or longer interval of waitlist risk could be considered, the risk of waitlist mortality at the three-month interval was assessed to allow for accurate comparisons to MELD score calculations.

26.4 Notes and Sources

This chapter is based on [41]. For background on liver transplantation see https://en.wikipedia.org/wiki/Liver_transplantation.

Chapter 27

The Final Word

Quote here.

– name here



10 pages

Introduction to the chapter here.

This is an introduction

References

- [1] Massie A.B., Caffo B., and et al. Gentry SE. Meld exceptions and rates of waiting list outcomes. *American Journal of Transplantation*, 11(11):2362–2371, 2011.
- [2] TS Arthanari and Yadolah Dodge. *Mathematical Programming in Statistics*, volume 341. Wiley, New York, 1981.
- [3] 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.
- [4] Susan Athey and Guido Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences*, 113(27):7353–7360, 2016.
- [5] 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.
- [6] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.
- [7] Kevin Bache and Moshe Lichman. UCI machine learning repository. <http://archive.ics.uci.edu/ml>, 2014.
- [8] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [9] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

- [10] 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.
- [11] Hamsa Bastani and Mohsen Bayati. Online decision-making with high-dimensional covariates. Available at SSRN 2661896, 2015. Working paper.
- [12] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton University Press, 2009.
- [13] A. Ben-Tal, E. Hazan, T. Koren, and S. Mannor. Oracle-based robust optimization via online learning. *Operations Research*, 63(3):628–638, 2015.
- [14] 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.
- [15] Kristin P Bennett and J Blue. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214, 1996.
- [16] 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.
- [17] E. R. Berndt and I. M. Cockburn. Price indexes for clinical trial research: a feasibility study. Technical report, National Bureau of Economic Research, 2013.
- [18] Dimitri P Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.
- [19] D. Bertsimas and N. Mundru. Sparse convex regression. *INFORMS Journal on Computing*, 2017. under review.
- [20] D. Bertsimas, D.B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011.
- [21] D. Bertsimas, M. Johnson, and N. Kallus. The power of optimization over randomization in designing experiments involving small samples. *Operations Research*, 63(4):868–876, 2015.
- [22] D. Bertsimas, J. Dunn, and N. Mundru. Optimal prescriptive trees. *INFORMS Journal on Optimization*, 2017. to appear.

- [23] D. Bertsimas, N. Korolko, and A. Weinstein. Identifying exceptional responders in randomized trials: An optimization approach. *INFORMS Journal on Optimization*, under review, 2018.
- [24] D. Bertsimas, R. Mazumder, and M. Sobiesk. Optimal classification and regression trees with hyperplanes are as powerful as classification and regression neural networks. *Journal of Machine Learning Research*, under review, 2018.
- [25] Dimitris Bertsimas and Martin S Copenhaver. Characterization of the equivalence of robustification and regularization in linear, median, and matrix regression. *European Journal of Operations Research*, 270: 931942, 2018.
- [26] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, pages 1–44, 2017.
- [27] Dimitris Bertsimas and Robert Freund. *Data, Models, And Decisions: The Fundamentals Of Management Science*. Dynamic Ideas Press, Belmont, Massachusetts, 2004.
- [28] Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *arXiv preprint arXiv:1402.5481*, 2014.
- [29] Dimitris Bertsimas and Nathan Kallus. Pricing from observational data. *arXiv preprint arXiv:1605.02347*, 2016.
- [30] Dimitris Bertsimas and Angela King. An algorithmic approach to linear regression. *Operations Research*, 64(1):2–16, 2016.
- [31] Dimitris Bertsimas and Angela King. Logistic regression: From art to science. *Statistical Science*, 32(3):367–384, 2017.
- [32] Dimitris Bertsimas and Michael Li. Accounting for significance and multicollinearity in building linear regression models. *INFORMS Journal on Optimization*, 2018. submitted for publication.
- [33] Dimitris Bertsimas and Rahul Mazumder. Least quantile regression via modern optimization. *The Annals of Statistics*, 42(6):2494–2525, 2014.
- [34] Dimitris Bertsimas and Romy Shioda. Classification and regression via integer optimization. *Operations Research*, 55(2):252–271, 2007.
- [35] Dimitris Bertsimas and Bart van Parys. Sparse high dimensional regression: Exact scalable algorithms and phase transitions. *Annals of Statistics*, 2016. submitted for publication.
- [36] Dimitris Bertsimas and Robert Weismantel. *Optimization over Integers*. Dynamic Ideas, Belmont, Massachusetts, 2005.

- [37] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *Annals of Statistics*, 44(2):813–852, 2016.
- [38] Dimitris Bertsimas, Angela King, Rahul Mazumder, et al. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, 2016.
- [39] Dimitris Bertsimas, Jack Dunn, Thomas A.Trikalinos, and Yuchen Wang. Improved triaging of diagnostic computed tomography for children after head trauma. *Pediatrics*, 2017. submitted for publication.
- [40] Dimitris Bertsimas, Nathan Kallus, Alex Weinstein, and Ying Daisy Zhuo. Personalized diabetes management using electronic medical records. *Diabetes Care*, 40(2):210–217, 2017.
- [41] Dimitris Bertsimas, Jerry Kung, Nikos Trichakis, Yuchen Wang, Ryutaro Hirose, and Parsia A. Vagefi. Optimized prediction of mortality (opom): a novel machine-learning approach for liver transplant allocation. *New England Journal of Medicine*, 2017. submitted for publication.
- [42] Dimitris Bertsimas, Jean Pauphilet, and Bart Van Parys. Sparse classification and phase transitions: a discrete optimization perspective. *Journal of Machine Learning Research*, 2017. Submitted for publication.
- [43] Dimitris Bertsimas, Colin Pawlowski, and Ying Zhuo. From predictive methods to missing data imputation: An optimization approach. *Journal of Machine Learning Research*, to appear, 2018.
- [44] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *arXiv preprint arXiv:1411.1607*, 2014.
- [45] S. Bezanson, J.and Karpinski, V. Shah, and A. Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.
- [46] Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.
- [47] Åke Björck. *Numerical methods for least squares problems*. SIAM, 1996.
- [48] Trond Hellem Bø, Bjarte Dysvik, and Inge Jonassen. LSimpute: accurate estimation of missing values in microarray data with least squares methods. *Nucleic Acids Research*, 32(3):e34–e34, 2004.

- [49] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.
- [50] 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.
- [51] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.
- [52] Lígia P Brás and José C Menezes. Improving cluster-based missing value estimation of DNA microarray data. *Biomolecular Engineering*, 24(2):273–282, 2007.
- [53] L. Breiman. Arcing the edge. Technical Report Technical Report 486, University of California, Berkeley, 1997.
- [54] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, California, 1984.
- [55] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [56] Leo Breiman. Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6):2350–2383, 1996.
- [57] Leo Breiman. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.
- [58] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [59] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199–231, 2001.
- [60] 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.
- [61] 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.
- [62] Dalia Buffery. The 2015 oncology drug pipeline: innovation drives the race to cure cancer. *American health & drug benefits*, 8(4):216, 2015.

- [63] Peter Bühlmann and Sara van-de-Geer. *Statistics for high-dimensional data*. Springer, 2011.
- [64] Lane F Burgette and Jerome P Reiter. Multiple imputation for missing data via sequential regression trees. *American Journal of Epidemiology*, 172:1070–1076, 2010.
- [65] Stef Buuren and Karin Groothuis-Oudshoorn. MICE: Multivariate imputation by chained equations in R. *Journal of Statistical Software*, 45(3), 2011.
- [66] D. Byar and S. Green. The choice of treatment for cancer patients based on covariate information. *Bulletin du cancer*, 67(4):477–490, 1979.
- [67] Zhipeng Cai, Maysam Heydari, and Guohui Lin. Iterated local least squares microarray missing value imputation. *Journal of Bioinformatics and Computational Biology*, 4(05):935–957, 2006.
- [68] C. Caramanis, S. Mannor, and H. Xu. *Optimization for machine learning*, chapter Robust optimization in machine learning. MIT Press, 2011.
- [69] R.J. Carroll, D. Ruppert, L.A. Stefanski, and C.M. Crainiceanu. *Measurement Error in Nonlinear Models: A Modern Perspective*. CRC Press, 2nd edition, 2006.
- [70] Rich Caruana. A non-parametric EM-style algorithm for imputing missing values. In *AISTATS*, 2001.
- [71] Probal Chaudhuri, Wen-Da Lo, Wei-Yin Loh, and Ching-Ching Yang. Generalized regression trees. *Statistica Sinica*, pages 641–666, 1995.
- [72] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754*, 2016.
- [73] K. Cho, D. Van Merriënboer, B. and Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.
- [74] 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.
- [75] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [76] 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.

- [77] G. B. Dantzig. Programming of interdependent activities: II mathematical model. *Econometrica*, 17:200–211, 1949.
- [78] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press and the RAND Corporation, 1963.
- [79] C. De Mol, E. De Vito, and L. Rosasco. Elastic-net regularization in learning theory. *Journal of Complexity*, 25(2):201–230, 2009.
- [80] Dean DeCock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3), 2011.
- [81] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (methodological)*, pages 1–38, 1977.
- [82] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.
- [83] Marcel Dettling. Bagboosting for tumor classification with gene expression data. *Bioinformatics*, 20(18):3583–3593, 2004.
- [84] D. Donoho and I. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81:425–455, 1993.
- [85] Jack Dunn. *Optimal Trees for Prediction and Prescription*,. PhD thesis, Massachusetts Institute of Technology, 2018.
- [86] 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.
- [87] Cholongitas E., Marelli L., and Kerry A. et al. Female liver transplant recipients with the same gfr as male recipients have lower meld scores-a systematic bias. *American Journal of Transplantation*, 7(3):685–692, 2007.
- [88] 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.
- [89] B. Efron and R. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

- [90] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, 1994.
- [91] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression (with discussion). *Annals of Statistics*, 32(2):407–499, 2004. ISSN 0090-5364.
- [92] Yonina C Eldar and Gitta Kutyniok. *Compressed sensing: theory and applications*. Cambridge University Press, London, 2012.
- [93] Sarah Elwir and John Lake. Current status of liver allocation in the united states. *Gastroenterology and Hepatology*, 12(3):166–170, 2016.
- [94] Saher Esmeir and Shaul Markovitch. Anytime learning of decision trees. *The Journal of Machine Learning Research*, 8:891–933, 2007.
- [95] M Faul, L Xu, M M Wald, and V G Coronado. *Traumatic brain injury in the United States: Emergency Department Visits, Hospitalizations and Deaths 20022006*. Centers for Disease Control and Prevention, National Center for Injury Prevention and Control, 2010.
- [96] 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.
- [97] R. A. Fisher. *The Design of Experiments*. Oliver and Boyd, Edinburgh, 1935.
- [98] T. Fleming and D. Harrington. *Counting Processes and Survival Analysis*, volume 169. John Wiley & Sons, 1991.
- [99] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1):327–349, 1994.
- [100] 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.
- [101] 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.
- [102] J. Foster, J. Taylor, and S. Ruberg. Subgroup identification from randomized clinical trial data. *Statistics in Medicine*, 30(24):2867–2880, 2011.

- [103] J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.
- [104] J. Friedman and N. Fisher. Bump hunting in high-dimensional data. *Statistics and Computing*, 9(2):123–143, 1999.
- [105] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.
- [106] J. Friedman, T. Hastie, and R. Tibshirani. GLMNet: Lasso and elastic-net regularized generalized linear models. r package version 1.9–5, 2013.
- [107] D. Gamarnik and I. Zadik. High-dimensional regression with binary coefficients. *arXiv preprint arXiv:1701.04455*, 2017.
- [108] 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.
- [109] M. Garey and D. Johnson. *A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
- [110] Zoubin Ghahramani and Michael I Jordan. Supervised learning from incomplete data via an EM approach. In *Advances in Neural Information Processing Systems*, pages 120–127, 1994.
- [111] L. El Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data. *SIAM Journal of Matrix Analysis and Applications*, 18(4):1035–64, 1997.
- [112] John C Gittins. *Multi-Armed Bandit Allocation Indices*. Wiley, Chichester, UK, 1989.
- [113] Alexander Goldenshluger and Assaf Zeevi. A linear response bandit problem. *Stochastic Systems*, 3(1):230–261, 2013.
- [114] G.H. Golub and C.F. Van Loan. An analysis of the total least squares problem. *SIAM Journal of Numerical Analysis*, 17(6):883–893, 1980.
- [115] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [116] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, Sh. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014.

- [117] I.J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
- [118] Google. Google search, 2018. URL https://scholar.google.com/scholar?q=neural+network+articles&btnG=&hl=en&as_sdt=0%2C22. Online; accessed 2018-01-11.
- [119] 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.
- [120] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 545–552. Curran Associates, Inc., 2009. URL <http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf>.
- [121] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.
- [122] R. Greevy, B. Lu, J. Silber, and Rosenbaum P. Optimal multivariate matching before randomization. *Biostatistics*, 5(2):263–275, 2004.
- [123] 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>.
- [124] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015. <http://www.gurobi.com>.
- [125] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015. <http://www.gurobi.com>.
- [126] Gurobi Optimization Inc. Gurobi 7.0 performance benchmarks. <http://www.gurobi.com/pdfs/benchmarks.pdf>, 2016. Accessed 17 December 2016.
- [127] D. Hardin, R. Rohwer, B. Curtis, A. Zagar, L. Chen, K. Boye, H. Jiang, and I. Lipkovich. Understanding heterogeneity in response to antidiabetes treatment: a post hoc analysis using sides, a subgroup identification algorithm. *Journal of Diabetes Science and Technology*, 7(2):420–430, 2013.

- [128] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: the Lasso and Generalizations*. CRC Press, 2015.
- [129] Trevor Hastie. Trevor hastie lectures and talks. http://www-stat.stanford.edu/~hastie/TALKS/glmnet_webinar_Rsession.tgz, 2015.
- [130] M.l Hay, J. Thomas, D.and Craighead, C. Economides, and J. Rosenthal. Clinical development success rates for investigational drugs. *Nature Biotechnology*, 32(1):40–51, 2014.
- [131] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [132] David Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *IJCAI*, pages 1002–1007. Citeseer, 1993.
- [133] David George Heath. *A geometric framework for machine learning*. PhD thesis, Johns Hopkins University, 1993.
- [134] Julie K. Heimbach, Ryutaro Hirose, and et al. Peter G. Stock. Delayed hepatocellular carcinoma model for end-stage liver disease exception score improves disparity in access to liver transplant in the united states. *Hepatology*, 61(5):1643–1650, 2015.
- [135] D. P. Helmbold and P. M. Long. On the inductive bias of dropout. *Journal of Machine Learning Research*, 16:3403–3454, 2015.
- [136] Harold V Henderson and Paul F Velleman. Building multiple regression models interactively. *Biometrics*, pages 391–411, 1981.
- [137] Jennifer L Hill. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, 2011.
- [138] J. Hodges and B. Michalowicz. Public-use data from the obstetrics and periodontal therapy (opt) study, a randomized trial of periodontal therapy to prevent pre-term birth. *Retrieved from the Data Repository for the University of Minnesota*, <http://dx.doi.org/10.13020/D6KW2D>, 2013.
- [139] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, pages 65–70, 1979.
- [140] James Honaker, Gary King, Matthew Blackwell, et al. Amelia II: A program for missing data. *Journal of Statistical Software*, 45(7):1–47, 2011.

- [141] R.A. Horn and C.R. Johnson. *Matrix analysis*. Cambridge University Press, second edition, 2013.
- [142] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: [http://dx.doi.org/10.1016/0893-6080\(89\)90020-8](http://dx.doi.org/10.1016/0893-6080(89)90020-8).
- [143] 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.
- [144] P.J. Huber and E.M. Ronchetti. *Robust statistics*. Wiley, second edition, 2009.
- [145] M. Hubert, P.J. Rousseeuw, and S. Van Aelst. High-breakdown robust multivariate methods. *Statistical Science*, 23(1):92–119, 2008.
- [146] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, 1976.
- [147] IBM ILOG CPLEX. V12.1 users manual, 2014.
- [148] 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.
- [149] Kosuke Imai and Marc Ratkovic. Covariate balancing propensity score. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(1):243–263, 2014.
- [150] 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.
- [151] Amir Jaffer and Lee Bragg. Practical tips for warfarin dosing and monitoring. *Cleveland Clinic journal of medicine*, 70(4):361–371, 2003.
- [152] V. Kaibel, M. Peinhardt, and M. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595–610, 2011.
- [153] Nathan Kallus. Recursive partitioning for personalization using observational data. In *International Conference on Machine Learning*, pages 1789–1798, 2017.
- [154] N. Karmarkar and R. Karp. Differencing method of set partitioning. Technical report, University of California, Berkeley, 1983.

- [155] N. Karmarkar, R. Karp, G. Lueker, and A. Odlyzko. Probabilistic analysis of optimum partitioning. *Journal of Applied Probability*, 23(3):626–645, 1986.
- [156] 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.
- [157] V. Kehl and K. Ulm. Responder identification in clinical trials with censored data. *Computational Statistics & Data Analysis*, 50(5):1338–1355, 2006.
- [158] James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.
- [159] Mohammad E Khan, Guillaume Bouchard, Kevin P Murphy, and Benjamin M Marlin. Variational bounds for mixed-data factor analysis. In *Advances in Neural Information Processing Systems*, pages 1108–1116, 2010.
- [160] 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.
- [161] Hyunsoo Kim, Gene H Golub, and Haesun Park. Missing value estimation for DNA microarray gene expression data: local least squares imputation. *Bioinformatics*, 21(2):187–198, 2005.
- [162] Ki-Yeol Kim, Byoung-Jin Kim, and Gwan-Su Yi. Reuse of imputed data in microarray analysis increases imputation efficiency. *BMC Bioinformatics*, 5(1):1, 2004.
- [163] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [164] 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.

- [165] A. Kurenkov. A 'brief' history of neural nets and deep learning, part 1, 2015. URL <http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/>. Online; accessed 2017-09-10.
- [166] Robert J LaLonde. Evaluating the econometric evaluations of training programs with experimental data. *The American economic review*, pages 604–620, 1986.
- [167] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: <http://dx.doi.org/10.1038/nature14539>.
- [168] Y. A. LeCun, L. Bottou, G. B. Orr, and K. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
- [169] 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.
- [170] 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/>.
- [171] M. Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>.
- [172] M. Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>.
- [173] 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.
- [174] C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region newton method for logistic regression. *Journal of Machine Learning Research*, 9(Apr):627–650, 2008.
- [175] 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.
- [176] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.
- [177] Roderick JA Little and Donald B Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, 1987.

- [178] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. John Wiley & Sons, 2014.
- [179] Wei-Yin Loh. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, pages 361–386, 2002.
- [180] Wei-Yin Loh and Yu-Shan Shih. Split selection methods for classification trees. *Statistica sinica*, pages 815–840, 1997.
- [181] Asdrúbal López-Chau, Jair Cervantes, Lourdes López-García, and Farid García Lamont. Fisher’s decision tree. *Expert Systems with Applications*, 40(16):6283–6291, 2013.
- [182] 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.
- [183] M. Lubin and I. Dunning. Computing in operations research using julia. *INFORMS Journal on Computing*, 27(2):238–248, 2015.
- [184] Shuangge Ma, Xiao Song, and Jian Huang. Supervised group lasso with applications to microarray data analysis. *BMC bioinformatics*, 8(1):60, 2007.
- [185] 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.
- [186] 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.
- [187] I. Markovsky and S. Van Huffel. Overview of total least-squares methods. *Signal Processing*, 87:2283–2302, 2007.
- [188] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of Machine Learning Research*, 11(Aug):2287–2322, 2010.
- [189] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115?133, 1943. doi: <https://link.springer.com/content/pdf/10.1007%2FBF02478259.pdf>.

- [190] S. Mertens. A physicist's approach to number partitioning. *Theoretical Computer Science*, 265(1):79–108, 2001.
- [191] 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.
- [192] T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.
- [193] Shakir Mohamed, Zoubin Ghahramani, and Katherine A Heller. Bayesian exponential family PCA. In *Advances in Neural Information Processing Systems*, pages 1089–1096, 2009.
- [194] K. Morgan and D. Rubin. Rerandomization to improve covariate balance in experiments. *The Annals of Statistics*, 40(2):1263–1282, 2012.
- [195] S. Morgenthaler. A survey of robust statistics. *Statistical Methods and Applications*, 15:271–293, 2007.
- [196] C. Morris. A finite selection model for experimental design of the health insurance study. *Journal of Econometrics*, 11(1):61–43, 1979.
- [197] S. Mosci, L. Rosasco, M. Santoro, A. Verri, and S. Villa. Solving structured sparsity regularization with proximal methods. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 418–433. Springer, 2010.
- [198] 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.
- [199] Sreerama Murthy and Steven Salzberg. Lookahead and pathology in decision tree induction. In *IJCAI*, pages 1025–1033. Citeseer, 1995.
- [200] Sreerama K Murthy and Steven Salzberg. Decision tree induction: How effective is the greedy heuristic? In *KDD*, pages 222–227, 1995.
- [201] 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.

- [202] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995.
- [203] 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.
- [204] Y. Nesterov. Gradient methods for minimizing composite objective function. Technical report, Center for Operations Research and Econometrics (CORE), Catholic University of Louvain, 2007. Technical Report number 76.
- [205] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer, Norwell, 2004.
- [206] 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.
- [207] Steven W Norton. Generating better decision trees. In *IJCAI*, volume 89, pages 800–805, 1989.
- [208] Shigeyuki Oba, Masa-aki Sato, Ichiro Takemasa, Morito Monden, Ken-ichi Matsubara, and Shin Ishii. A Bayesian missing value estimation method for gene expression profile data. *Bioinformatics*, 19(16):2088–2096, 2003.
- [209] Ziad Obermeyer and Ezekiel J. Emanuel. Predicting the future - big data, machine learning, and clinical medicine. *The New England Journal of Medicine*, 375(13):1216, 2016.
- [210] 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.
- [211] Rosenbaum P. and D. Rubin. Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. *The American Statistician*, 30(1):33–38, 1985.
- [212] 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.

- [213] 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*. NIHR Journals Library, 2011.
- [214] Mahesh KB Parmar, James Carpenter, and Matthew R Sydes. More multiarm randomised trials of superiority are needed. *The Lancet*, 384(9940):283, 2014.
- [215] Harold J Payne and William S Meisel. An algorithm for constructing optimal binary decision trees. *Computers, IEEE Transactions on*, 100(9):905–916, 1977.
- [216] 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.
- [217] 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.
- [218] Min Qian and Susan A Murphy. Performance guarantees for individualized treatment rules. *Annals of statistics*, 39(2):1180, 2011.
- [219] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [220] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.
- [221] John R Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. Singapore, 1992.
- [222] Wiesner R., E. Edwards, and Freeman R. et. al. Model for end-stage liver disease (meld) and allocation of donor livers. *Gastroenterology*, 124(1):91–96, 2003.
- [223] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014. URL <http://www.R-project.org/>.
- [224] 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/>.

- [225] Trivellore E Raghunathan, James M Lepkowski, John Van Hoewyk, and Peter Solenberger. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology*, 27(1):85–96, 2001.
- [226] S. Raschka. Single-layer neural networks and gradient descent, 2015. URL http://sebastianraschka.com/Articles/2015_singlenet_neurons.html. Online; accessed 2017-09-9.
- [227] G. Raskutti, M. J. Wainwright, and B. Yu. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *Journal of Machine Learning Research*, 15(1):335–366, 2014.
- [228] Soo-Yon Rhee, Jonathan Taylor, Gauhar Wadhera, Asa Ben-Hur, Douglas L Brutlag, and Robert W Shafer. Genotypic predictors of human immunodeficiency virus type 1 drug resistance. *Proceedings of the National Academy of Sciences*, 103(46):17355–17360, 2006.
- [229] P. Rosenbaum and D. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.
- [230] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton project para. Technical report, Cornell Aeronautical Laboratory, 1957.
- [231] P.J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association*, 79:871–80, 1984.
- [232] D. Rubin. Using multivariate matched sampling and regression adjustment to control bias in observational studies. *Journal of the American Statistics Association*, 74(366):318–328, 1979.
- [233] D. Rubin. Comment: Which ifs have causal answers. *Journal of the American Statistical Association*, 81(366):961–962, 1986.
- [234] Da. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL <http://dl.acm.org/citation.cfm?id=65669.104451>.
- [235] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL <http://arxiv.org/abs/1409.0575>.
- [236] Maura E Ryan, Susan Palassis, 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 traumachild. *Journal of the American College of Radiology*, 11(10):939–947, 2014.
- [237] Eddington S. and Onghena P. *Randomization Tests, Fourth Ed.* CRC press, 2007.
 - [238] M. Schemper. Non-parametric analysis of treatment-covariate interaction in the presence of censoring. *Statistics in Medicine*, 7 (12):1257–1266, 1988.
 - [239] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
 - [240] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2001.
 - [241] B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT press, 2002.
 - [242] U. Shaham, Y. Yamada, and S. Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.
 - [243] R. Socher, B. Huval, B. Bhat, C. D. Manning, and A. Y. Ng. Convolutional-Recursive Deep Learning for 3D Object Classification. In *Advances in Neural Information Processing Systems 25*. 2012.
 - [244] Nguyen Hung Son. From optimal hyperplanes to optimal decision trees. *Fundamenta Informaticae*, 34(1, 2):145–174, 1998.
 - [245] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.
 - [246] Daniel J Stekhoven and Peter Bühlmann. Missforest: non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012.
 - [247] X. Su, CL. Tsai, H. Wang, D. Nickerson, and B. Li. Subgroup analysis via recursive partitioning. *Journal of Machine Learning Research*, 10 (Feb):141–158, 2009.
 - [248] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

- Curran Associates, Inc., 2014. URL <http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf>.
- [249] Tensorflow. URL <https://opensource.google.com/projects/tensorflow>.
- [250] 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.
- [251] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.
- [252] A.N. Tikhonov. On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198, 1943.
- [253] 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.
- [254] Top500 Supercomputer Sites. Performance development. <http://www.top500.org/statistics/perfdevel/>, 2016. Accessed 17 December 2016.
- [255] Luís Torgo. Regression datasets. <http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>, 2014.
- [256] Luis Torgo. Regression data sets. <http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>, 2017. Accessed 9 November 2017.
- [257] Luís Fernando Raíño Alves Torgo. *Inductive learning of tree-based regression models*. PhD thesis, Universidade do Porto, 1999.
- [258] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.
- [259] Alfred Truong. *Fast growing and interpretable oblique trees via logistic regression models*. PhD thesis, University of Oxford, 2009.
- [260] Athanasios Tsanas and Angeliki Xifara. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings*, 49:560–567, 2012.
- [261] Panagiotis Tzirakis and Christos Tjortjis. T3c: improving a decision tree classification algorithms interval splits on continuous attributes. *Advances in Data Analysis and Classification*, pages 1–18, 2016.

- [262] U.S. Food and Drug Administration. *The Drug Development Process: Clinical Research*, 2015. <http://www.fda.gov/ForPatients/Approvals/Drugs/ucm405622.htm>.
- [263] Stef Van Buuren. Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research*, 16(3):219–242, 2007.
- [264] V. Vapnik. The support vector method of function estimation. In *Nonlinear Modeling*, pages 55–85. Springer, 1998.
- [265] S. Wager and S. Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 2017. to appear.
- [266] M.J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using-constrained quadratic programming (Lasso). *IEEE Transactions on Information Theory*, 55(5):2183–2202, 2009.
- [267] Xian Wang, Ao Li, Zhaohui Jiang, and Huanqing Feng. Missing value estimation for DNA microarray gene expression data by support vector regression imputation and orthogonal coding scheme. *BMC Bioinformatics*, 7(1):1, 2006.
- [268] P. Werbos. *Beyond regression : new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.
- [269] 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.
- [270] T. Wheldon. *Mathematical Models in Cancer Research*. Adam Hilger, 1988.
- [271] DC Wickramarachchi, BL Robertson, M Reale, CJ Price, and J Brown. Hhcrt: An oblique decision tree. *Computational Statistics & Data Analysis*, 96:12–23, 2016.
- [272] Larry Winner. Miscellaneous datasets. <http://www.stat.ufl.edu/~winner/datasets.html>, 2014.
- [273] Larry Winner. Miscellaneous data sets. <http://www.stat.ufl.edu/~winner/datasets.html>, 2017. Accessed 9 November 2017.
- [274] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.
- [275] H. Xu, C. Caramanis, and S. Mannor. Robust regression and Lasso. *IEEE Transactions in Information Theory*, 56(7):3561–74, 2010.

- [276] I-Cheng Yeh. Modeling of strength of high performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12):1797–1808, 1998.
- [277] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.
- [278] 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.
- [279] F. Zhang. *The Schur complement and its applications*, volume 4. Springer, 2006.
- [280] Xiaobai Zhang, Xiaofeng Song, Huinan Wang, and Huanping Zhang. Sequential local least squares imputation estimating missing value of microarray data. *Computers in Biology and Medicine*, 38(10):1112–1120, 2008.
- [281] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. *Facial Landmark Detection by Deep Multi-task Learning*. Springer International Publishing, 2014. URL https://doi.org/10.1007/978-3-319-10599-4_7.
- [282] 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.
- [283] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B*, 67(2):301–320, 2005.