Analytics under Variability, Volume, and Velocity with Applications to Sustainability and Healthcare

by

Vasileios Digalakis

Diploma in Electrical and Computer Engineering,
Technical University of Crete, Greece (2018)

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

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Author .....
Sloan School of Management
May 5, 2023

Certified by .....
Dimitris Bertsimas
Boeing Leaders for Global Operations Professor of Management
Thesis Supervisor

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

Analytics under Variability, Volume, and Velocity with Applications to Sustainability and Healthcare

by

Vasileios Digalakis

Submitted to the Sloan School of Management
on May 5, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

Analytics, machine learning, and optimization provide unique opportunities to harness the massive amounts of data that are available and positively impact some of the most pressing challenges of our time, including climate change and improved healthcare operations. The classical paradigm of analytics, which assumes a dataset is centrally collected and readily available to analyze, is shifting. Modern data science problems present new complexities, including variability (i.e., changing phenomena due to various types of uncertainties), large volumes of data or decisions or both, and data arriving dynamically with high velocity.

This thesis advances two strands of large-scale analytics. The first is methodological, focusing on the development of predictive and prescriptive machine learning and optimization methodologies, primarily mixed-integer and robust, for problems that exhibit the aforementioned characteristics. The second is applied, and encompasses collaborations with various industry partners in the sustainability and healthcare operations spaces, seeking to reap the benefits of large-scale analytics in these settings.

In Chapters 2 and 3, we introduce the framework of slowly varying machine learning, which provides a tool to deal with variability in an interpretable way. In Chapter 2 in particular, our methodology enables the estimation of sparse linear regression models where the underlying regression coefficients are allowed to vary slowly and sparsely under some graph-based temporal or spatial structure. In Chapter 3, we take a step toward the stabilization of decision tree models even under new trends in the training data. In Chapter 4, we introduce the backbone method, a general, heuristic framework that scales interpretable machine learning techniques to ultra-high dimensional datasets hence tackling the volume characteristic. Chapter 5 develops a mixed integer optimization- and machine learning-based approach for the problem of frequency estimation in data streams, addressing settings where large amounts of data arrive dynamically with high velocity. Finally, in Chapter 6, we present a robust optimization- and machine learning-based framework that guides a 1 billion USD investment in solar panels and batteries by a leading fertilizer producer, with the

aim of decarbonizing a significant portion of their production pipeline and reducing operational costs. Our model's forecast indicates that this decarbonization effort will be profitable, thus emphasizing that investing in renewable energy can be a financially viable option, rather than an expensive luxury that developing nations cannot afford while industrializing their economies.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Leaders for Global Operations Professor of Management

Acknowledgments

In gratitude, I wish to extend my acknowledgments to those who have been part of this journey and aided it in ways beyond measure.

Dimitris Bertsimas and I both bore mustaches as we embarked on our respective doctoral journeys. Yet as we reached their conclusions, we both had relinquished these hairy companions. Fortunately, his influence on me has extended far, far, far beyond facial hair, shaping not only my approach to research but also my philosophy towards life. For, as an advisor, he has guided me in navigating the ways of academia and in optimally deciding what to do with the time that was given to us. For, as a mentor, he has taught me that the worth of research lies not in the intricacy of the mathematics nor the complexity of the theories, but in the positive change it brings to the world. For, as a friend, he has instilled within me the significance of approaching research and life with confidence and optimism, particularly when the road darkens.

In the guidance of Alexandre Jacquillat, I have grown to become a much, much-improved researcher and communicator of my research, and beyond that, he has also aided me in bettering my skills as a tennis player and runner. Wolfram Wiesemann and I have shared many inspiring discussions over the past year, and his diligent and insightful commentary has significantly enhanced my work. Conversing with Nikos Trichakis has been a source of inspiration too; his advice on my work and my next steps has been invaluable, and his progress in mastering the art of tennis has been remarkable. Rama Ramakrishnan has profoundly influenced my teaching approach after I served as his teaching assistant. Jónas Oddur Jónasson, Daniel Freund, Rahul Mazumder, Bart Van Parys, and Patrick Jaillet have provided precious feedback on my work or have taught me informative classes or both. Georgia Perakis, Patrick, and Dimitris through their leadership, and Laura Rose, Andrew Carvalho, and Gloria Adduci through their tireless efforts, have ensured that the Operations Research Center is one of the most exceptional academic environments.

To this academic realm of exceptional excellence, I owe much gratitude for the extraordinary scholarly interactions. Michael Lingzhi Li, Ryan Cory-Wright, and

Yu Ma have been more than mere collaborators, offering steadfast dedication and brilliant insights. Theodore Papalexopoulos have helped me cogitate through nearly every decision I have made, academic and beyond. My dear friend of old, Yannis Spantidakis, together with Aristeidis Karalis and Patricio Foncea Araneda have kept the music ringing — watching, waiting, commiserating. Agni Orfanoudaki, Ilias Zadik, and Alvaro Fernandez Galiana have inspired me beyond words and have served as my mentors of greatness. Jean Pauphilet, Arthur Delarue, and Holly Wiberg have been most generous in sharing their wisdom with me. Moïse Blanchard, Samuel Gilmour, and Shuvomoy Das Gupta have been astute conversants over my scholarship. Alongside unparalleled exchanges of intellect, the arduous journey has been enriched by the dear friends I have met along the way.

The fellowship of the Greeks has always welcomed me back home, often kept vigil through the night conversing with me through the telephone, and some even dared to cross the great ocean to visit me. Zaha has kept watch with tireless eyes and overseen my labors, as the greater portion of this work was underway, and even now that this very piece is being written. Areti has had a great hand in shaping who I am today; her accomplishments have been great, and I deem there shall be many more to come — do not give in without a fight! Francine is guiding me through wondrous paths of life, and much do I owe her for the fulfillment of this journey, and for the radiance that the future holds — spring has come!

My parents, Vassilis and Vana, have been an unwavering source of support, love, and encouragement, and have transformed me, once among the world's most impatient, into a more persevering and determined person. Now they have set out on a perilous quest to transform the ailing Greek educational system. My sister, Korina, has lent her ear to my endless grievances about the ways of mathematics, the trials of tennis, and the challenges of life itself. Her brilliance has been an unattainable standard and her example an ever-present inspiration; perhaps only in the courts of tennis may I stand equal, and even then, I count myself fortunate. It is to the latter three that I humbly dedicate this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Global Covenant of Mayors for Climate and Energy (GCoM) [1] is an international alliance of cities and local governments committed to taking action on climate change and, more generally, driving progress towards a more sustainable future. The initiative uses big data to track greenhouse gas emissions and energy consumption in participating cities, and provides tools and resources to help these cities reduce their carbon footprint. As of 2021, the initiative has over 10,000 signatories from around the world, representing over 1.2 billion people.

Participating cities are required to report their greenhouse gas emissions using a standardized methodology. Moreover, they are encouraged to collect data on their energy consumption, including electricity, heating, and transportation fuels; their vulnerability to climate change impacts, such as sea level rise, extreme weather events, and drought; and, finally, public health data, such as air quality, water quality, and access to healthcare services, to better understand the sustainability-healthcare interface and the health impacts of climate change. The data submitted by participating cities can be accessed through the initiative's data portal (https://dataportalforcities.org/).

GSoM is one of the numerous policy initiatives that rely on data and analytics to combat climate change and promote sustainability. It also perfectly exemplifies a large-scale problem that challenges conventional analytics techniques: as the initiative expands, the *volume* of data will explode; the *velocity* at which data arrives may increase too; and, perhaps most importantly, the problem is subject to *variability*,

owing to the presence of different types of uncertainties.

This thesis develops predictive and prescriptive analytics methodologies that address the aforementioned challenges in impactful, real-world applications in the sustainability and healthcare spaces. For example, on the *predictive* side, we introduce the framework of slowly varying regression under sparsity primarily through the lens of an energy consumption prediction case study. By enabling the predictive model to vary slowly depending on the hour of the day or the period of the year (hence dealing with variability), the model's predictive performance and interpretability both improve (see Chapter 2). On the *prescriptive* side, we develop an optimization framework to guide the intended 1 billion USD investment in solar panels and batteries by one of the world's largest fertilizer producers. By guarding the prescriptive model against uncertainty in solar generation and reducing the (initially large) problem dimension, the prescribed solution improves in terms of both cost savings and reduction in carbon emissions (see Chapter 6).

In the rest of the introduction, we define the complexities (variability, volume, velocity) addressed by our methodological contributions (Section 1.1), explain the concept of variability and its sources (Section 1.2), list our main contributions by chapter (Section 1.3), and discuss recent advances in interpretable machine learning models that are central to this thesis (Section 1.4).

1.1 The V-Complexity of Modern Datasets

"There is nothing permanent except change."

Heraclitus

The preceding discussion suggests that the classical paradigm of data analytics, which assumes a dataset is centrally collected and readily available to analyze, is shifting. Many data science problems are characterized by the so-called V-complexity [133, 146]:

- *Variability*: the problem exhibits changing phenomena owing to various types of uncertainties.

- *Volume*: the problem (number of decisions) or the data (number of data points or features or both) or both are large.
- *Velocity*: the data arrive dynamically in the form of a large stream and need to be processed in real-time.

Chapters 2, 3, and 6 in this thesis focus on variability, while Chapters 4 and 6 deal with volume, and Chapter 5 tackles velocity. By addressing these complexities, we aim to push the frontier of analytics to problems of larger sizes, under changing phenomena and uncertainties, or with dynamically arriving data.

1.2 Variability: From Known Knowns to Known Unknowns and Unknown Unknowns

> *"There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know."*
>
> Donald Rumsfeld
> Former US Secretary of Defense

The term "variability" is used as an overarching concept for various types of changing phenomena that may be present in a data science problem. Temporal variability refers to changes over time, such as daily fluctuations in energy consumption due to the time of day or season. Spatial variability refers to changes across different locations, like variations in air quality measurements across different parts of a city. Pattern variability refers to changes in the underlying patterns or structures, such as periodic (seasonal) sales patterns. Chapter 2 covers problems with all three types of variability, while Chapters 3 and 6 focus on temporal variability.

Variability can arise due to different reasons, including known knowns, known unknowns, and unknown unknowns, all of which describe the amount and type of uncertainty in the problem [181, 113]. *Known knowns* are the factors that are already known and available to the model, such as the training data. Variability due to known knowns occurs when the underlying changing phenomena are predictable and understood. For instance, the seasonal changes in temperature are known knowns, and their variability can be modeled and predicted.

Known unknowns are the factors that are known to exist but not fully quantified. Variability due to known unknowns occurs when the source of the underlying changing phenomena is understood, but their magnitude or impact is uncertain. In our energy consumption prediction case study discussed in Chapter 2, we recognize that the daily weather and human behavior patterns are potential sources of variability for the predictive model, but the precise nature of their impact is yet to be quantified. In Chapter 6, the variability in solar capacity factors due to changing weather patterns throughout the day is another example of known unknowns. Climate change also constitutes a known unknown, as the exact degree to which it will affect weather patterns remains uncertain.

Unknown unknowns are the factors that are not known to exist and cannot be anticipated or predicted. Variability due to unknown unknowns occurs when the source and magnitude of the underlying changing phenomena are both unknown and unpredictable. Examples of unknown unknowns in machine learning include new trends or sudden changes in the underlying population — both of which are common in healthcare settings, where more data becomes available over time. In Chapter 3, we focus on improving stability in decision tree models to address this type of uncertainty; decision trees are highly susceptible to changes in the training data, and our work aims to make them more stable against sudden changes in the underlying population. Chapter 6 provides another example of unknown unknowns, as we explore the variability in solar capacity factors due to abrupt climate shifts and extreme weather events, which may be triggered by unforeseen events such as the collapse of a major ice sheet.

1.3 Contributions and Outline

"The Road goes ever on and on down from the door where it began. Now far ahead the Road has gone, and I must follow, if I can."

J.R.R. Tolkien

Overall, this thesis seeks to advance two strands of large-scale analytics. The first is methodological, focusing on the development of machine learning (ML) and optimization methodologies, primarily mixed-integer (MIO) and robust (RO), for big-data problems with a large V-complexity. MIO, in particular, can naturally represent interpretable ML models, such as sparse regression and decision trees, and often improves their predictive power and proves their optimality in fitting the training data; scaling such methodologies to problems with a large V-complexity is therefore of paramount importance. The second is applied, and encompasses collaborations with various industry partners in the sustainability and healthcare operations spaces, seeking to reap the benefits of large-scale analytics in these settings.

The summary and main contributions of this thesis follow, listed by chapter. To simplify the presentation, we defer all technical proofs and extended numerical experiments to the end of each chapter.

Chapters 2 and 3 Variability and Slowly Varying ML Models

The *variability* characteristic appears in numerous data science applications, ranging from sustainability and healthcare to politics and finance.

As discussed in Section 1.2, variability can be due to known unknowns: the source of the variability is understood, but the magnitude is uncertain. For example, the important features for predicting a building's energy consumption may change depending on the hour of the day or period of the year, and are likely to do so in a "continuous" way over time; the same holds true for meteorological models or

voting patterns in the US electorate over adjacent spatial areas, and factor models in the stock market over consecutive time periods. In Chapter 2, we introduce the framework of slowly varying regression under sparsity, which provides a tool to deal with such applications in an interpretable way. Our methodology enables the rigorous estimation of sparse linear regression models where the underlying regression coefficients are allowed to vary slowly and sparsely under some graph-based temporal or spatial structure. Our computational study suggests that our algorithm outperforms competing formulations across a variety of metrics, on both synthetic and real-world data, including applications in energy consumption prediction, air quality monitoring, and meteorology; we also deploy our method in an industry collaboration in preventive maintenance [43].

Our formulation minimizes the total prediction error across all time periods or spatial areas, and includes both standard regularization considerations (for robustness and sparsity purposes) and new ones (to prevent sharp variations in coefficients). Our main theoretical contribution is an exact binary convex reformulation of the problem using a new equality on Moore-Penrose inverses that convexifies the non-convex objective function while coinciding with the original objective on all feasible binary points; this approach also extends to a more general class of sparse quadratic problems. From an algorithmic standpoint, we provide a cutting plane-type algorithm that solves the reformulated convex problem to optimality, and propose ways to leverage its structure to further improve its efficiency. We also develop a fast, heuristic alternative that produces feasible, high-quality solutions, and make our implementation available open-source for use by practitioners.

Variability can also be due to unknown unknowns, when both its source and magnitude are unknown. For example, consider a healthcare setting where we build an interpretable ML model to support a physician's decision-making. A commonly encountered situation is that the initial dataset is small but larger amounts of data become available over time as more patients' information gets recorded. Then, it is reasonable to consider retraining the ML model to boost its predictive performance and incorporate new patterns in the data — which were unknown unknowns when the

original model was trained. Upon retraining, the new model changes notably, owing to the instability of the underlying ML algorithm. This situation harms interpretability and hinders the adoption of the model by the physician. In Chapter 3, we take a step towards the stabilization of decision tree models through the lens of real-world healthcare applications.

We introduce a new distance measure for decision trees and use it to determine a tree's level of stability. We propose a novel methodology to train stable decision trees and investigate the existence of trade-offs that are inherent to decision tree models — including between stability, predictive power, and interpretability. We demonstrate the value of the proposed methodology through an extensive quantitative and qualitative analysis of six case studies from the healthcare space due to the relevance of stability and interpretability in this space, and we show that, on average, with a small 4.6% decrease in predictive power, we gain a significant improvement in the model's stability by as much as 38%.

The material in this chapters is based on two papers. [44] is completed and currently under second round of review in Operations Research (Major Revision). [42] is completed and currently under first round of review in Nature Machine Intelligence.

Chapter 4: Volume and the Backbone Method

Applications in genomics, medical imaging, satellite imagery, and finance, among others, commonly exhibit the *volume* characteristic, involving ultra-high dimensional datasets with millions of features. Over the past decade, MIO-based methods for interpretable supervised ML, e.g., sparse regression, have scaled from being intractable beyond few hundreds of features to now being able to handle hundreds of thousands. In Chapter 4, we introduce the backbone method, a general, heuristic framework to scale such techniques to ultra-high dimensional datasets; of note, our method solves, in minutes, sparse regression problems with tens of millions of features, and decision tree problems with hundreds of thousands of features. We show, on both synthetic and real-world datasets, that our method outperforms or competes with state-of-the-art

methods, without sacrificing the interpretability of the learned model.

The proposed framework operates in two phases: we first extract a "backbone set" of potentially relevant features by solving a number of specially-chosen, tractable subproblems; we then use traditional techniques to solve a reduced problem to optimality or near-optimality, considering only the backbone features. We develop backbone methods for various supervised ML models, including sparse linear and logistic regression, and classification trees. For the case of sparse linear regression, our theoretical analysis shows that, under certain conditions and with high probability, the backbone set consists only of the truly relevant features.

This chapter is based on [41], which has appeared in Machine Learning.

Chapter 5: Velocity and Frequency Estimation in Data Streams

The *velocity* characteristic of modern datasets is ubiquitous in streaming settings, where massive amounts of data arrive dynamically and must be processed in real-time, e.g., finding trending topics by counting queries to a search engine. Since the stream of queries (and even its frequency distribution) is too large to store, state-of-the-art techniques typically rely on random hashing to combine frequencies for different queries and so approximate the full distribution. In Chapter 5, we propose an MIO- and ML-based approach to hashing in this setting, learning patterns in the initial part of the stream that help avoid costly collisions between frequencies. Our method outperforms existing approaches by one-to-two orders of magnitude in approximation error per query on real-world search query data.

The contributions of our work are two-fold. From a technical standpoint, we develop exact (MIO- and dynamic programming-based) and heuristic (block coordinate descent) algorithms to learn an optimal (or near-optimal) hashing scheme from queries that appeared in the observed stream prefix, and use ML to hash unseen queries. From a conceptual standpoint, we show how MIO and ML can be of value in streaming settings, where their use has been scarce.

This chapter is based on [40], which has appeared in IEEE Transaction in Knowledge

and Data Engineering.

Chapter 6: Solar Capacity Expansion at OCP under Variability and Volume

In Chapter 6, we present our collaboration with OCP, a phosphate mining and fertilizer production company that accounts for 5.6% of Morocco's GDP, to help them fulfill their pledge of being carbon neutral by 2040. Excluding cogeneration, nearly 90% of OCP's energy demand is currently satisfied via carbon-emitting sources. Our modeling suggests that strategic placement of solar panels and batteries can reduce their dependence on non-green electricity purchased from the Moroccan grid by as much as 80%, while also decreasing their long-term operational costs by over $2.5Bn. OCP senior management are now using our code, in collaboration with our team, to size their investment.

We develop a multi-period optimization framework that guides OCP's strategic decisions (when and where to install solar panels and batteries) and operational decisions (how to operate their system given the strategic decisions they have made so far) over a 20-year horizon. To tackle the large dimensionality (*volume*) of the problem, we propose a novel, ML-based scenario reduction technique, which allows us to plan for a reduced set of "typical days" rather than all days in the planning horizon. To guard against uncertainty in forecast solar generation (*variability*), we combine RO with distributionally robust optimization (DRO). We use RO with a custom, data-driven uncertainty set and a novel constraint aggregation technique to prevent conservatism. We implement DRO techniques to protect against climate shifts. The end result is a flexible and highly scalable framework for strategic decision-making.

This chapter is based on [36], which is completed and currently under second round of review in M&SOM (Major Revision) as well as a second-round contender in the 2023 M&SOM Practice-Based Research Competition, a biennial competition for the M&SOM journal.

1.4 Background

We briefly review the landscape of the sparse regression and the decision tree problem, which we shall commonly revisit throughout this thesis. In terms of notation, we are given a set of independent identically distributed data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^p$ and responses $y_1, \ldots, y_n \in \mathbb{R}$, and our goal is to train a predictive model (sparse regression or decision tree) $f : \mathbb{R}^p \mapsto \mathbb{R}$ such that the model output $f(\boldsymbol{x}_i)$ accurately estimates the response y_i.

Advances in Sparse Regression

A regression problem is considered *high dimensional* when the rank of the data matrix $\boldsymbol{X}$ is smaller than the number of features p, i.e., $\text{rank}(\boldsymbol{X}) < p$; this is, for example, the case when n, the number of data points in the dataset, satisfies $n < p$. In such regimes, the regression model is challenged, as the underlying linear system is underdetermined, and further assumptions are required. The application of sparse or sparsity-inducing methods provides a way around this limitation and, in addition, hopefully, leads to more interpretable models, where only a few features actually affect the prediction.

Exact Sparse Regression Formulation. The sparse regression problem with an explicit sparsity constraint, also known as the best subset selection problem, is the most natural way of performing simultaneous feature selection and model fitting for regression. The sparsity constraint is imposed by requiring that the ℓ_0 (pseudo)norm of the vector of regressors $\boldsymbol{w} \in \mathbb{R}^p$ is less than a predetermined degree of sparsity $k \ll p$, namely, $\|\boldsymbol{w}\|_0 \leq k$, where $\|\boldsymbol{w}\|_0 = \sum_{j=1}^{p} \mathbf{1}_{(w_j \neq 0)}$ and $\mathbf{1}_{(\cdot)}$ denotes the indicator function. However, the ℓ_0 constraint is very far from being convex, and the resulting combinatorial optimization problem is NP-hard [173].

Motivated by the advances in MIO and despite the diagnosed hardness of the problem, a recent line of work solves the problem exactly to optimality or near-optimality (i.e., within a user-specified optimality gap that can be set to an arbitrarily small value up to machine precision). [34] cast best subset selection as a mixed integer

quadratic program and solve it with a commercial solver for problems with $p \sim 10^3$. [35] reformulate the sparse regression problem as a pure binary optimization problem and use a cutting planes-type algorithm that, based on the outer approximation method of [93], iteratively tightens a piece-wise linear lower approximation of the objective function. By doing so, they solve to optimality sparse regression problems with $p \sim 10^5$. [44] consider a variant of the sparse regression problem, namely, slowly varying sparse regression, and develop a highly optimized version of the outer approximation method by utilizing a novel convex relaxation of the objective function; their technique can be directly applied to the standard sparse regression problem and scale it to larger instances. [131] develop a specialized, nonlinear branch and bound framework that exploits the structure of the sparse regression problem (e.g., they use tailored heuristics to solve node relaxations) and are able to solve problems with $p \sim 8 \cdot 10^6$.

From a practical point of view, one argument commonly used against best subset selection is that, in real-world problems, the actual support size k is not known and needs to be thoroughly cross-validated hence resulting in a dramatic increase in the required computational effort. Although, in many cases, k is determined by the application, [143] address such concerns by proposing efficient cross-validation strategies.

Heuristic Solution Methods and Surrogate Formulations. Traditionally, the exact sparse regression formulation has been addressed via heuristic and greedy procedures (dating back to [23, 96, 136]). Recently, numerous more sophisticated methods have been proposed, including the boolean relaxation of [186], the first-order method of [34], the subgradient method of [35] and [50], the method of [132] that combines coordinate descent with local search, and the approach of [61] that is based on the alternating direction method of multipliers.

A lot of effort has been dedicated to developing and solving surrogate problems; the most studied of them all is the lasso formulation [210], whereby the non-convex ℓ_0 norm is replaced by the convex ℓ_1 norm. Namely, denoting by $\|\boldsymbol{w}\|_1 = \sum_{j=1}^{p} |w_j|$,

we require that $\|\boldsymbol{w}\|_1 \leq t$ or add a penalty term $\lambda\|\boldsymbol{w}\|_1$ in the objective. Due to convexity, the constrained and the penalized versions can be shown to be equivalent for properly selected values of t and λ; this is not the case if the ℓ_0 norm is used. Due to the geometry of the unit ℓ_1 ball, the resulting formulations indeed shrink the coefficients toward zero and produce sparse solutions by setting many coefficients to be exactly zero. There has been a substantial amount of algorithmic work on the lasso formulation [95, 25, 109] and its many variants (e.g., the elastic net formulation of [240]), on replacing the ℓ_1 norm in the Lasso formulation by other sparsity inducing penalties (e.g., [99, 234]), and on developing scalable implementations [110].

Advances in Decision Trees

Decision trees are among the most popular and interpretable methods in machine learning. At a high level, decision trees recursively partition the feature space into disjoint regions and assign to each resulting partition either a label, in the context of classification, or a constant or linear prediction, in the context of regression. The leading work for decision tree methods is the classification and regression trees framework (CART), proposed by [63], which takes a top-down approach to determine the partitions. Briefly, the CART method operates in two phases:

- A top-down induction phase, where, starting from a root node, a split is determined by minimizing an "impurity measure" (e.g., entropy), and then recursively applying the process to each of the resulting child nodes until no more splits are possible.

- A pruning phase, where the learned tree is pruned to penalize more complex structures that might not generalize well.

The tree induction process of CART has several limitations. First, the process is one-step optimal and not overall optimal. Second, it does not directly optimize over the actual objective (e.g., misclassification error). Third, it only considers univariate splits, i.e., splits that involve a single feature. Despite of the aforementioned limitations,

CART have been widely used as building blocks in ensemble learning methods based on bagging and boosting; examples include the random subspace method [135], random forest [62], and XGBoost [79]. Indeed, such methods enhance the performance of the resulting classifier, sacrificing, however, the interpretability of the model.

[45] formulate the decision tree problem using MIO and propose a tailored coordinate descent-based solution method. The resulting optimal trees framework (OT) overcomes many of the limitations of CART (optimization is over the actual objective, and multivariate splits are possible), while still leading to highly interpretable models. The key observation that led to OT is that, when learning a decision tree, there is a number of discrete decisions and outcomes we need to consider:

- Whether to split at any node and which feature to split on.

- Which leaf node a point falls into and (for classification problems) whether this point is correctly classified based on its label.

At each branch node, a split of the form $\boldsymbol{a}^T\boldsymbol{x} < b$ is applied. Points that satisfy this constraint follow the left branch of the tree, whereas those that violate the constraint follow the right branch. Each leaf node is assigned a label, and each point is assigned the label of the leaf node into which the point falls. The resulting MIO formulation aims to make the aforementioned decisions in such a way that the linear combination of an error metric (e.g., misclassification error) and a tree complexity measure is minimized. The formulation also includes a number of constraints ensuring that the resulting tree is indeed valid and consistent. The resulting MIO formulation is solved using a tailored coordinate descent approach; the problem is nonconvex, so the solution process is repeated from a variety of starting trees that are generated randomly and, in the end, the one with the lowest overall objective function is selected. A long stream of literature has followed the original work by [45] in trying to optimally or near-optimally solve the decision tree problem, including, e.g., the work by [7], who develop a flow-based MIO formulation.

Chapter 2

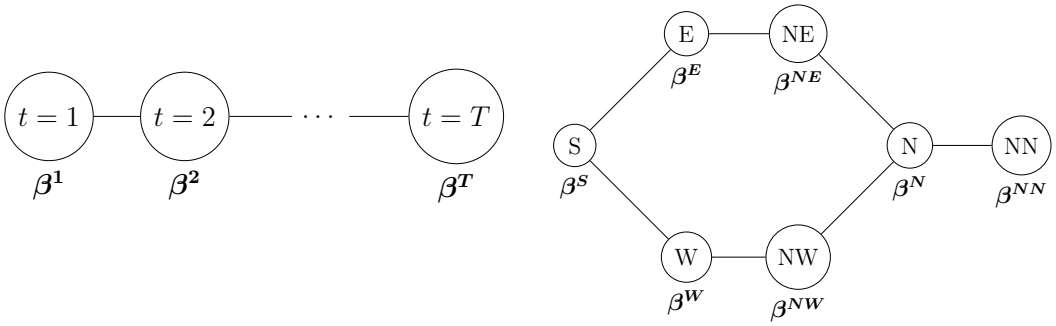
Variability and Slowly Varying Regression under Sparsity

2.1 Introduction

We introduce the framework of *slowly varying regression under sparsity*, which addresses a large number of problems in machine learning where the underlying model is sparse and varies slowly and sparsely. This in particular includes problems with *temporally or spatially varying structure*. For example, in the temporal case, the factors important in predicting the energy consumption in a building can vary depending on the hour of the day or the period of the year. In the spatial case, the factors that affect house prices can differ by neighborhoods. In both cases, a modeler may be motivated to use different models for each time period, spatial area, or, more generally, "vertex." However, using separate models ignores the innate dependence across different vertices (e.g., energy consumption is strongly affected by daily and seasonal patterns) and creates interpretability issues if the trained separate models turn out to be notably different. Further, separate models require substantial data to be available for each vertex, which is often difficult. In this work, we address the need to capture this slowly and sparsely varying structure in a global way.

2.1.1 An Initial Formulation

Formally, we consider a multiple regression problem with N cases having features $\boldsymbol{X}^1, \ldots, \boldsymbol{X}^T$, where $\boldsymbol{X}^t \in \mathbb{R}^{N \times D}$ for $t \in [T] := \{1, \ldots, T\}$, and outcomes $\boldsymbol{y}^1, \ldots, \boldsymbol{y}^T$, where $\boldsymbol{y}^t \in \mathbb{R}^N$ for $t \in [T]$. The idea of slowly varying regression under sparsity (SSVR) assumes that the regression coefficients and relevant features (i.e., features that correspond to nonzero coefficients) have to change slowly between pairs of regressions $(s, t) \in [T] \times [T]$ that are considered *similar*. Two of the most prominent applications include temporally varying regression and spatially varying regression. In the *temporal* case, the regressions are scattered over T consecutive time periods, and regressions between two consecutive time periods are considered to be similar. In the *spatial* case, the regressions are conducted over T spatial areas, some of which are adjacent to each other, and it is common to assume that regressions in adjacent areas have to be similar. More generally, the T regressions are conducted over a graph G with vertices V of size $|V| = T$. For $v, w \in V$, the edge (v, w) is in the set of edges E if and only if v and w are considered to be similar. Figure 2-1 presents examples of similarity graphs.



(a) Temporal similarity graph.

(b) Spatial similarity graph.

Figure 2-1: Examples of similarity graphs. In the temporal case (left), the different regressions are applied across T consecutive time periods. (Such examples are considered in Section 2.8.2.) In the spatial case (right), the different regressions are applied across 7 spatial areas (S, E, W, NE, NW, N, NN) with the given similarity structure. (The graph corresponds to one of the experiments described in Section 2.8.3.)

The SSVR problem can be formulated as below:

$$\min_{\boldsymbol{\beta}^1,\cdots,\boldsymbol{\beta}^T} \quad \sum_{t=1}^{T} \left\|\boldsymbol{y}^t - \boldsymbol{X}^t\boldsymbol{\beta}^t\right\|_2^2 + \lambda_\beta \sum_{t=1}^{T} \|\boldsymbol{\beta}^t\|_2^2 + \lambda_\delta \sum_{(s,t)\in E} \|\boldsymbol{\beta}^t - \boldsymbol{\beta}^s\|_2^2 \tag{2.1}$$

$$\text{s.t.} \quad \left|\text{Supp}(\boldsymbol{\beta}^t)\right| \leq K_{\text{L}}, \quad \forall t \in [T], \tag{2.2}$$

$$\left|\bigcup_{t=1}^{T} \text{Supp}(\boldsymbol{\beta}^t)\right| \leq K_{\text{G}}, \tag{2.3}$$

$$\sum_{(s,t)\in E} \left|\text{Supp}(\boldsymbol{\beta}^t) \triangle \text{Supp}(\boldsymbol{\beta}^s)\right| \leq K_{\text{C}}, \tag{2.4}$$

where $\text{Supp}(\boldsymbol{\beta})$ denotes the set that corresponds to the support of vector $\boldsymbol{\beta}$ and $S_1 \triangle S_2$ denotes the symmetric difference of sets S_1, S_2. The objective function (2.1) penalizes both the least-squares loss of the T regressions and the ℓ_2 coefficient distance between regressions that are similar with magnitude λ_δ. We also introduce a further ℓ_2 regularization term of magnitude λ_β for robustness purposes (see, e.g., [228]). There are three types of constraints on the regression coefficients $\boldsymbol{\beta}^t$:

- *Local Sparsity*: Each regression has at most K_{L} relevant features (constraint (2.2)).

- *Global Sparsity*: There are at most K_{G} relevant features across all T regressions (constraint (2.3)).

- *Sparsely Varying Support*: There is a difference of at most K_{C} relevant features among similar regressions s, t across all pairs of similar regressions (constraint (2.4)).

For consistency, $K_{\text{L}}, K_{\text{G}}, K_{\text{C}}$ satisfy $K_{\text{L}} \leq K_{\text{G}} \leq D$ and $K_{\text{C}} \leq 2K_{\text{L}}T$ and $2(K_{\text{G}} - K_{\text{L}}) \leq K_{\text{C}}$. This exact formulation is generally considered infeasible beyond toy scales ($D \leq 10^2,\ T \leq 10$) due to the combinatorial complexity of the sparsity constraints. Therefore, many authors have proposed various relaxations in order to solve variants of this problem, including fused lasso [211] and sum-of-norms regularization [178]; we review such approaches in Section 2.1.2. Our key contribution in this chapter is to show that this general problem can be reformulated as a binary convex optimization

problem, which then can be solved efficiently using a cutting plane-type algorithm. This reformulation is primarily enabled by an exact smooth relaxation of the solution under sparsity constraints, which, to the best of our knowledge, has not appeared in prior literature. Furthermore, we discuss in Section 2.1.3 how the reformulation directly extends to any sparse quadratic convex problem of a general form, making the relaxation generally applicable.

2.1.2 Relevant Literature: Slowly Varying Regression

The practical relevance of the notion of *slowly (or smoothly) varying regression* is evident by the significant amount of impactful work in the field, dating back to at least [129], who study linear regression models whose coefficients are allowed to change smoothly with the value of other variable, and [32], who solve the nonparametric regression estimation problem when the underlying regression function is Lipschitz continuous; see, e.g., the book by [98] for a comprehensive review and e.g. [185, 76] for applications in Econometrics and Electronics.

The popularity of slowly varying regression models peaked following the work of [211] on the *fused lasso*, which proposed to augment the standard lasso [210] objective with an ℓ_1 penalty term on the difference between successive regression coefficients $|\beta^t - \beta^{t-1}|$ to account for pairwise similarity. The algorithm for solving the resulting problems were improved by many works including [212, 225] while other works considered different convex regularizers or extend the formulation to other settings such as change point detection [12, 192, 58].

The works that are most closely related to ours include the sum-of-norms regularization approach by [178] and total variation regularization approach by [224]. [178] considers the time-varying linear regression optimization problem, and their work can naturally be extended to the (more general) graph case that we consider in this chapter as follows:

$$\min_{\boldsymbol{\beta}^1,\cdots,\boldsymbol{\beta}^T} \quad \sum_{t=1}^{T} \left\|\boldsymbol{y}^t - \boldsymbol{X}^t\boldsymbol{\beta}^t\right\|_2^2 + \lambda_\delta \sum_{(s,t)\in E} \|\boldsymbol{\beta}^t - \boldsymbol{\beta}^s\|_p, \tag{2.5}$$

where $p \in \{1, 2\}$. Our work significantly differs with this line of work by exactly imposing sparsity and sparse variation in the coefficients, hence providing more control to the modeler, and, further, by utilizing a smooth ℓ_2 penalty term on the difference between the regression coefficients $\|\boldsymbol{\beta}^t - \boldsymbol{\beta}^s\|_2^2$.

Another stream of related work in the application of *spatially varying regression* are spatially varying coefficient (SVC) models. Instead of imposing a strict constraint on the degree of variability, SVC models focus on identifying the heterogeneity in coefficient estimates varying across space. Notable methods include the spatial expansion method [73], geographically weighted regression [69], and Bayesian SVC models [55].

2.1.3 Relevant Literature: Solving Sparse Quadratic Models

Problems with sparsity constraints have long been of interest in many areas ranging from machine learning to facility location and portfolio selection, as sparsity improves robustness to data noise and increases interpretability for better decision-making. However, due to their combinatorial complexity, it has long been thought that exact sparse formulations are not scalable, and ℓ_1 regularization formulations (e.g. fused lasso in Section 2.1.2) have been widely used as surrogates.

In recent years, a growing volume of work challenges the aforementioned paradigm. [34, 35, 131] solve the standard sparse regression problem with design matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, responses $\boldsymbol{y} \in \mathbb{R}^N$, and ℓ_2 regularization, outlined as

$$\min_{\boldsymbol{\beta}:\|\boldsymbol{\beta}\|_0 \leq K} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda_\beta \|\boldsymbol{\beta}\|_2^2, \tag{2.6}$$

at scale, using techniques from mixed-integer optimization. The work that is most closely related to ours is by [35], who utilize binary variables $\boldsymbol{z} \in \{0, 1\}^D$ and reformulate Problem (2.6) as

$$\min_{\boldsymbol{z} \in \{0,1\}^D, \sum_{i=1}^D z_i \leq K} \min_{\boldsymbol{\beta}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{Z}\boldsymbol{\beta}\|_2^2 + \lambda_\beta \|\boldsymbol{\beta}\|_2^2, \tag{2.7}$$

where $\boldsymbol{Z} = \text{Diag}(z_1, \ldots, z_D)$. The authors then show that the inner minimization problem can be solved in a closed form that results in a convex binary formulation for the outer problem:

$$\min_{\boldsymbol{z}\in\{0,1\}^D, \sum_{i=1}^D z_i \leq K} \boldsymbol{y}^T \left(\boldsymbol{I}_N + \frac{1}{2\lambda_\beta} \sum_{i=1}^D z_i \boldsymbol{X}_i \boldsymbol{X}_i^T \right)^{-1} \boldsymbol{y}, \tag{2.8}$$

where $\boldsymbol{X}_i$ is the ith column of design matrix $\boldsymbol{X}$. The resulting problem can then be solved efficiently to very large scales ($N, D \approx 10^4$) using a cutting plane-type algorithm. This significant breakthrough raised the limits of scaling exact sparse methods by multiple orders of magnitude.

However, the transformation presented above seems to be quite fortuitous. For example, the reformulation in [35] relied on rewriting $\boldsymbol{\beta}$ as $\boldsymbol{Z}\boldsymbol{\beta}$ in the first term but not the second term of Equation (2.7). There appears to be no systematic reason why doing so is necessary to result in the final convex binary formulation in Equation (2.8), and even fewer hints on how we could systematically apply this methodology to other problems.

This chapter aims to uncover the underlying key ingredients to allow such transformations through the study of the SSVR problem, which is a generalization of the standard sparse regression problem of Equation (2.6). Moreover, the framework we present here directly extends to any sparse quadratic convex problem of the form

$$\min_{\boldsymbol{x}\in\mathcal{X}} \quad \frac{1}{2}\boldsymbol{x}^T(\boldsymbol{M} + \lambda \boldsymbol{I})\boldsymbol{x} - \boldsymbol{\mu}^T\boldsymbol{x}, \tag{2.9}$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$, $\boldsymbol{x} \in \mathbb{R}^D$ are the decision variables, $\boldsymbol{M} \in \mathbb{R}^{D\times D}$ is any positive semidefinite matrix, and $\mathcal{X}$ is the feasible set with sparsity-imposing constraints, e.g. $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{R}^D : \|\boldsymbol{x}\|_0 \leq K\}$.

Overall, the proposed framework lies between [35] and [37]. *On one extreme*, we consider a general quadratic regression framework that contains the sparse regression problem in [35] as a special case and draw insights from optimization and convex relaxations to efficiently solve the resulting inner problem. *On the other extreme*, [37]

investigate general sparse mixed-integer optimization problems with logical constraints and develop an outer approximation scheme whereby the solution to the inner problem involves solving an optimization problem in each iteration; in contrast, we focus on sparse mixed-integer optimization problems where the inner problem is an unconstrained quadratic optimization problem and develop a general efficient procedure for optimizing the resulting problem.

2.1.4 Contributions and Outline

We now more concretely summarize our contributions from a modeling, a theoretical, an algorithmic, and a computational (practitioner's) perspective:

- From a *modeling* standpoint, we introduce the SSVR framework, which addresses regression problems with sparse and slowly varying structure.

- From a *theoretical* standpoint, we propose a new way of solving the underlying optimization problem, which extends to a more general class of sparse quadratic problems. We reformulate the problem exactly as a binary convex optimization problem through a novel exact relaxation of the objective function. The proposed relaxation relies upon a new equality on Moore-Penrose inverses that convexifies the nonconvex objective function while coinciding on all feasible binary points.

- From an *algorithmic* standpoint, leveraging the convexity of the reformulated problem, we develop a cutting plane-type algorithm that enables us to solve the binary convex optimization problem at hand to provable optimality. By exploiting the structure of the problem, we efficiently implement the proposed algorithm and substantially improve upon the asymptotic computational complexity of a straightforward implementation. Further, we develop a fast heuristic algorithm, which is guaranteed to produce a feasible solution and, as we empirically show, computes high-quality, warm-start solutions to the binary convex optimization problem.

- From a *computational* standpoint, we thoroughly evaluate the proposed method

on both synthetic and real-world data. We show that the proposed algorithm outperforms competing formulations across a variety of metrics including estimation accuracy, predictive power, and computational time, and is highly scalable, enabling us to train models with 10,000s of parameters. In real-world experiments, we further illustrate how the resulting SSVR model can provide insights into the problem at hand. We make our implementation available *open-source* at `https://github.com/vvdigalakis/SSVRegression.git`. To facilitate the use of the proposed framework by practitioners, all proposed algorithms can be run through a single line of code, and the learned models are provided in an intuitive and interpretable way.

The rest of the chapter is organized as follows. In Section 2.2, we formulate the SSVR problem as a mixed-integer optimization problem. In Section 2.3, we reformulate the problem exactly as a binary convex optimization problem through the novel exact relaxation of the original problem that we develop. Section 2.4 explores the properties of the proposed relaxation, builds intuition on why it works, and studies how it can be extended to general quadratic models. We then develop the proposed exact cutting plane type algorithm and discuss how to efficiently implement it (Section 2.5), as well as the proposed fast heuristic algorithm (Section 2.6). Finally, Sections 2.7 and 2.8 present our experimental evaluations on synthetic and real-world data, respectively.

2.2 An MIO Formulation

In this section, we develop a mixed-integer optimization (MIO) formulation for the problem defined in (2.1)–(2.4). To do so, we encode each of the aforementioned constraints using auxiliary binary variables and new constraints.

Local Sparsity. First, we introduce binary variables $\boldsymbol{z}^t$ encoding the support of the coefficients $\boldsymbol{\beta}^t,\ \forall t$, as $z_d^t = 0 \Rightarrow \beta_d^t = 0,\ \forall t \in [T], d \in [D]$. Then the requirement that the number of nonzero coefficients at each vertex is less than K_{L} can be expressed as $\sum_{d=1}^{D} z_d^t \leq K_{\mathrm{L}},\ \forall t \in [T]$.

Global Sparsity. Similarly, we introduce binary variables encoding the union of supports over vertices. We require that s_d is set to 1 if z_d^t is set to 1 at least once over all vertices, i.e., $s_d \geq z_d^t,\ \forall t \in [T], d \in [D]$. then we have $\sum_{d=1}^{D} s_d \leq K_{\mathrm{G}}$.

Sparsely Varying Support. To be able to capture the sparsely varying support requirement, we introduce another set of binary variables $w_d^{t,s} = 0 \Rightarrow \|\beta_d^t\|_0 = \|\beta_d^s\|_0 \Rightarrow z_d^t = z_d^s,\ \forall (s,t) \in E, d \in [D]$. This can be rewritten as $w_d^{t,s} \geq z_d^t - z_d^s$ and $w_d^{t,s} \geq z_d^s - z_d^t,\ \forall (s,t) \in E, d \in [D]$. We then require that $\sum_{(s,t)\in E} \sum_{d=1}^{D} w_d^{t,s} \leq K_{\mathrm{C}}$.

Overall Formulation. With these helper binary variables and constraints, we can now rewrite the original problem defined in (2.1)–(2.4) as follows:

$$\min_{\substack{\boldsymbol{z}\in\{0,1\}^{TD},\\ \boldsymbol{s}\in\{0,1\}^{D},\\ \boldsymbol{w}\in\{0,1\}^{|E|D}}} \ \min_{\boldsymbol{\beta}} \quad \sum_{t=1}^{T} \left\|\boldsymbol{y}^t - \boldsymbol{X}^t \boldsymbol{Z}^t \boldsymbol{\beta}^t\right\|_2^2 + \lambda_\beta \sum_{t=1}^{T} \|\boldsymbol{Z}^t \boldsymbol{\beta}^t\|_2^2 + \lambda_\delta \sum_{(s,t)\in E} \|\boldsymbol{Z}^t \boldsymbol{\beta}^t - \boldsymbol{Z}^s \boldsymbol{\beta}^s\|_2^2 \tag{2.10}$$

$$\text{s.t.} \quad \sum_{d=1}^{D} z_d^t \leq K_{\mathrm{L}}, \qquad \forall t \in [T], \tag{2.11}$$

$$s_d \geq z_d^t, \qquad \forall t \in [T], d \in [D], \tag{2.12}$$

$$\sum_{d=1}^{D} s_d \leq K_{\mathrm{G}}, \tag{2.13}$$

$$w_d^{t,s} \geq z_d^t - z_d^s, \qquad \forall (s,t) \in E, d \in [D], \tag{2.14}$$

$$w_d^{t,s} \geq z_d^s - z_d^t, \qquad \forall (s,t) \in E, d \in [D], \tag{2.15}$$

$$\sum_{(s,t)\in E} \sum_{d=1}^{D} w_d^{t,s} \leq K_{\mathrm{C}}, \tag{2.16}$$

where $\boldsymbol{Z}^t = \mathrm{Diag}(z_1^t, \cdots, z_D^t)$ are diagonal binary matrices of $\boldsymbol{z}$ variables. For convenience, we denote the optimization problem over $\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w}$ as the outer optimization problem, while the optimization over $\boldsymbol{\beta}$ as the inner optimization problem.

2.3 The Binary Convex Reformulation

In this section, we reformulate the mixed-integer optimization problem defined in (2.10)-(2.16) as a pure-binary convex optimization problem. First, we note the following lemma:

Lemma 1. *The MIO optimization problem defined in* (2.10)-(2.16) *is equivalent to the following optimization problem:*

$$\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}} \min_{\boldsymbol{\beta}} \quad c(\boldsymbol{z},\boldsymbol{\beta}) := \frac{1}{2}\boldsymbol{\beta}^\top(\boldsymbol{Z}(\boldsymbol{M}+\lambda_\beta \boldsymbol{I})\boldsymbol{Z})\boldsymbol{\beta} - \boldsymbol{\mu}^\top \boldsymbol{Z}\boldsymbol{\beta},$$

where $\boldsymbol{\beta} = (\boldsymbol{\beta}^1, \cdots, \boldsymbol{\beta}^T)$, $\boldsymbol{Z} = \mathrm{Diag}(\boldsymbol{z}^1, \cdots, \boldsymbol{z}^T)$, *and* $\mathcal{Z}$ *is the polyhedral feasible set as defined by the binary constraints on* $\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w}$ *and* (2.11)-(2.16). $\boldsymbol{M} \in \mathbb{R}^{TD\times TD}$ *and* $\boldsymbol{\mu} = (\boldsymbol{\mu}^1, \cdots, \boldsymbol{\mu}^T)$ *are defined as:*

$$\begin{aligned}
\boldsymbol{M}^{t,s}_{i,j} = &\left[\sum_n \left(X^t_{n,i}\right)^2 \;+\; d^t\lambda_\delta\right] \mathbb{1}_{(s=t \text{ and } i=j)} \;+\; \left[\sum_n X^t_{n,i}X^t_{n,j}\right] \mathbb{1}_{(s=t \text{ and } i\neq j)} \\
&-\; \lambda_\delta \; \mathbb{1}_{((s,t)\in E \text{ and } i=j)}, \\
\boldsymbol{\mu}^t = &(\boldsymbol{X}^t)^\top \boldsymbol{y}^t,
\end{aligned}$$

where d^t *denotes the degree of vertex* t. *Furthermore,* $\boldsymbol{M}$ *is a positive semi-definite matrix.*

The proof is given in Section 2.9. With this formulation, we can solve the inner problem easily using the first order condition and reduce the problem to a binary optimization problem. Recall that the Moore-Penrose pseudoinverse $\boldsymbol{A}^\dagger \in \mathbb{R}^{n\times m}$ of $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is the unique matrix that satisfies the four conditions: 1. $\boldsymbol{A}^\dagger\boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger$, 2. $\boldsymbol{A}\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{A}$, 3. $(\boldsymbol{A}\boldsymbol{A}^\dagger)^* = \boldsymbol{A}\boldsymbol{A}^\dagger$, 4. $(\boldsymbol{A}^\dagger\boldsymbol{A})^* = \boldsymbol{A}^\dagger\boldsymbol{A}$, where $*$ is the Hermitian operator with $\boldsymbol{A}^*_{ij} = \overline{\boldsymbol{A}_{ji}}$. We can then prove:

Lemma 2. *Denote* $\boldsymbol{\beta}^*(\boldsymbol{z}) = \mathit{argmin}_{\boldsymbol{\beta}} c(\boldsymbol{z},\boldsymbol{\beta})$. *Then we have*

$$\boldsymbol{\beta}^*(\boldsymbol{z}) = (\boldsymbol{Z}(\boldsymbol{M}+\lambda_\beta\boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z}\boldsymbol{\mu}. \tag{2.17}$$

Furthermore, it holds that

$$\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}} \min_{\boldsymbol{\beta}} c(\boldsymbol{z},\boldsymbol{\beta}) = \min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}} -\frac{\boldsymbol{\mu}^\top \boldsymbol{\beta}^*(\boldsymbol{z})}{2}.$$

The proof is given in Section 2.9. Unfortunately, the resulting formulation $\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}} -\frac{\boldsymbol{\mu}^\top \boldsymbol{\beta}^*(\boldsymbol{z})}{2}$ is neither convex nor differentiable in $\boldsymbol{z}$, when $\boldsymbol{\beta}^*(\boldsymbol{z}) = (\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z}\boldsymbol{\mu}$, making the problem intractable. However, we have the following key observation: we only care about $\boldsymbol{\beta}^*(\boldsymbol{z})$ for binary vectors $\boldsymbol{z}$. Therefore, we proceed to consider exact convex relaxations of $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z}$ such that it agrees with $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z}$ on all binary points $\boldsymbol{z}$. Specifically, we prove the following proposition, which allows us to convexify the expression above:

Proposition 1. *Let $\boldsymbol{M}$ be a positive semi-definite matrix. Then we have, for $\boldsymbol{z} \in \{0,1\}^{TD}$, $\boldsymbol{Z} = \mathrm{Diag}(\boldsymbol{z}^1, \cdots, \boldsymbol{z}^T)$, and $\lambda_\beta > 0$:*

$$(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z}. \tag{2.18}$$

The proof is included in Section 2.9. Finally, we prove that, using the reformulation in Proposition 1, the problem becomes convex in $\boldsymbol{z}$:

Theorem 1. *Let $\boldsymbol{M}, \boldsymbol{\mu}$ be defined in Lemma 1, and $\lambda_\beta > 0$. Then the optimization problem in (2.10)–(2.16) is equivalent to the following binary convex optimization problem:*

$$\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}} -\frac{\boldsymbol{\mu}^\top \boldsymbol{\beta}^*(\boldsymbol{z})}{2}, \tag{2.19}$$

where $\boldsymbol{\beta}^*(\boldsymbol{z}) = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z}\boldsymbol{\mu}$.

The proof is given in Section 2.9. Theorem 1 shows that the original problem as shown in (2.10)-(2.16) can be reformulated into a binary convex optimization problem over $\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w}$, which is amenable to a cutting plane-type algorithm. We point out that the key ingredient that enabled such convex relaxation, Proposition 1, is by no means obvious: there are infinitely many relaxations that match exactly the binary points of $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z}$. In fact, an arguably more natural construction of a relaxation is

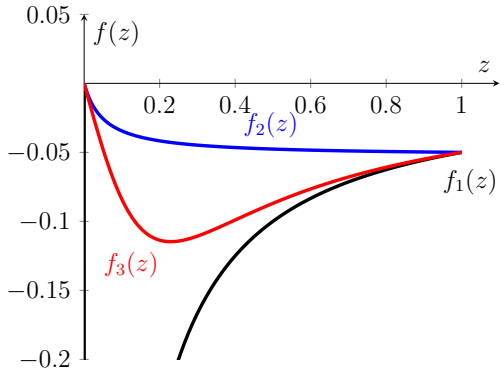


Figure 2-2: Various relaxations of the pseudoinverse.

the following equality (that can be easily shown using Lemma 6):

$$(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{Z}. \tag{2.20}$$

However, such a relaxation, unlike the one shown in Proposition 1, results in a *non-convex* reformulation of the problem as stated in (2.10)-(2.16), making it significantly more difficult to solve.

2.4 Discussion of the Relaxation

As shown in Section 2.3, the key observation that enabled us to create the convex reformulation is Proposition 1. In this section, we discuss why the relaxation works and illustrate it intuitively.

For simplicity, we consider the case where we have $T = D = 1$, and a single binary variable z. Then by Theorem 1, the objective function for the optimization problem defined in (2.10)–(2.16) has the form:

$$f_1(z) = -(z(m + \lambda_\beta)z)^\dagger \mu^2 z = \begin{cases} 0, & z = 0, \\ -\frac{\mu^2}{z(m+\lambda_\beta)}, & z \neq 0. \end{cases}$$

Proposition 1 then reads, for all $m > 0$ and $z \in \{0, 1\}$, $(z(m + \lambda_\beta)z)^\dagger z = \frac{z}{\lambda_\beta + mz}$. After reformulation, the objective function has the form $f_2(z) = -\frac{\mu^2 z}{\lambda_\beta + mz}$. While

the other natural relaxation we can construct, as defined in Equation (2.20) gives the objective function $f_3(z) = -\frac{\mu^2 z}{\lambda_\beta + mz^2}$. In Figure 2-2 we plot $f_1(z), f_2(z), f_3(z)$ for $m = 19$, $\mu = 1$, and $\lambda_\beta = 1$. First we observe that in one dimension, the pseudoinverse is a discontinuous and non-convex function that follows a $-\frac{1}{z}$ type curve everywhere except for $z = 0$, where it takes the value of 0. This clearly reflects the difficulty to solve the sparse problem as formulated in the standard way.

We then observe that both $f_2(z)$ and $f_3(z)$ agree with $f_1(z)$ when $z \in \{0, 1\}$, and therefore $f_2(z)$, $f_3(z)$ are both valid relaxations of the discontinuous function $f_1(z)$ on the binary values of z. However, we clearly see that $f_2(z)$ is a convex function in z, while $f_3(z)$ is not. This illustrates how the carefully chosen relaxation enables efficient convex algorithms to be utilized.

We finally note that the exact relaxation utilized in this chapter, $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z}$, is not the only exact convex relaxation possible. For example, the following formula extends our relaxation into a family of (exact) relaxations for all $\mu \geq 0$:

$$(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z} + \mu \left(\sum_{t=1}^{T} \sum_{d=1}^{D} \left(z_d^t - \frac{1}{2} \right)^2 - \frac{TD}{4} \right),$$

However, experiments on synthetic data (see Section 2.10.4) suggest that algorithm performance in both solution time and accuracy decays as μ increases, so our original relaxation is superior in performance. We believe the development of more effective convex (exact) relaxations is a fruitful direction for future research.

2.5 An Exact Cutting Plane Algorithm

In this section, we propose a cutting plane-type algorithm that solves Problem (2.19) to optimality. The proposed Algorithm 1 is based on the outer approximation method by [93], which iteratively tightens a piecewise linear lower approximation of the objective function. Algorithm 1 provides pseudocode for the proposed approach.

Recall from Theorem 1 that the objective function $c(\boldsymbol{z}, \boldsymbol{\beta})$ is indeed convex in $\boldsymbol{z}$

Algorithm 1: Cutting Plane Algorithm

Input: Data $(\boldsymbol{X}^t, \boldsymbol{y}^t)_{t=1}^T$, similarity graph G, sparsity parameters $(K_{\mathrm{L}}, K_{\mathrm{G}}, K_{\mathrm{C}})$, regularization parameters $(\lambda_\beta, \lambda_\delta)$.

Output: Learned coefficients $\boldsymbol{\beta}^\star$.

▷ `Find warm start using Algorithm 2:`

$\boldsymbol{\beta}^{(0)} \leftarrow$ `find_start` $\left((\boldsymbol{X}^t, \boldsymbol{y}^t)_{t=1}^T, G, (K_{\mathrm{L}}, K_{\mathrm{G}}, K_{\mathrm{C}}), (\lambda_\beta, \lambda_\delta)\right)$

▷ `Compute corresponding binary variables:`

$(\boldsymbol{z}^{(0)}, \boldsymbol{s}^{(0)}, \boldsymbol{w}^{(0)}) \leftarrow$ `find_binaries` $\left(\boldsymbol{\beta}^{(0)}\right)$

$(i, \eta^{(0)}) \leftarrow (0, 0)$

▷ `Cutting plane iterations:`

while $c(\boldsymbol{z}^{(i)}) > \eta^{(i)}$ **do**

$(\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w}, \eta)^{(i+1)} \leftarrow \underset{\substack{\eta \in \mathbb{R}_+, \\ \boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w} \in \mathcal{Z}}}{\operatorname{argmin}} \eta$ s.t. $\eta \geq c(\boldsymbol{z}^{(\tau)}) + \nabla_{\boldsymbol{z}} c(\boldsymbol{z}^{(\tau)})^\top (\boldsymbol{z} - \boldsymbol{z}^{(\tau)}),\ \forall \tau \in [i]$

$i \leftarrow i + 1$

end while

▷ `Estimate coefficients using Theorem 1:`

$\boldsymbol{\beta}^\star \leftarrow \boldsymbol{\beta}^*(\boldsymbol{z}^{(i)})$

return $\boldsymbol{\beta}^\star$

and can, in fact, be written as function only of the binary variables $\boldsymbol{z}$ by solving the inner problem to optimality, i.e.,

$$\min_{\boldsymbol{\beta}} c(\boldsymbol{z}, \boldsymbol{\beta}) := c(\boldsymbol{z}) = -\frac{\boldsymbol{\mu}^\top \boldsymbol{\beta}^*(\boldsymbol{z})}{2} = -\frac{1}{2} \boldsymbol{\mu}^\top (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} \boldsymbol{Z}\boldsymbol{\mu}. \tag{2.21}$$

Algorithm 1 also requires the computation of the gradient of the cost function $\nabla_{\boldsymbol{z}} c(\boldsymbol{z})$ at every binary point $\boldsymbol{z}$ it visits. We therefore aim to differentiate the loss function $c(\boldsymbol{z})$ with respect to the diagonal entries of the matrix $\boldsymbol{Z} = \mathrm{Diag}(\boldsymbol{z}^1, \ldots, \boldsymbol{z}^T)$. The partial derivative with respect to component z_d^t can be computed numerically using finite differences as $\frac{\partial c(\boldsymbol{z})}{\partial z_d^t} = \frac{c(\boldsymbol{z}) - c(\boldsymbol{z} - \varepsilon \boldsymbol{e_d^t})}{\varepsilon}$, where $\boldsymbol{e_d^t}$ denotes the basis vector with 1 in position (t, d) and 0's elsewhere and ε is a sufficiently small constant. Such an approach would be highly impractical, as it would require TD evaluations of the cost function (2.21). Instead, we utilize the chain rule to compute the gradient in closed form, as shown below:

Lemma 3. *Let* $\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{z}) := (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})$ *and let* $\boldsymbol{E_d^t}$ *denote a* $TD \times TD$ *matrix, with* 1 *at position* $(t, d), (t, d)$ *and* 0*'s elsewhere. Then we have:* $\frac{\partial c(\boldsymbol{z})}{\partial z_d^t} =$

$\frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{K}^{-1} \left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}} \boldsymbol{M} \boldsymbol{K}^{-1} \boldsymbol{Z} \; - \; \boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\right) \boldsymbol{\mu}$.

The proof is given in Section 2.9. We next discuss the computational complexity of the cut generation for Algorithm 1. Recall the cut generation process requires the evaluation of the cost function $c(\boldsymbol{z})$ and its gradient $\nabla_{\boldsymbol{z}} c(\boldsymbol{z})$. Lemma 4 enables us to generate cuts more efficiently than we would with a naive implementation.

Lemma 4. *Let $\boldsymbol{z}$ be a feasible binary vector for Problem* (2.19)*. Then the cost function $c(\boldsymbol{z})$ and its gradient $\nabla_{\boldsymbol{z}} c(\boldsymbol{z})$ can be evaluated in $O\left(T^3 K_L^2 + T^2 K_L^3 + T^2 K_L D\right)$ operations.*

The proof is given in Section 2.9. Finally, Theorem 2 asserts that Algorithm 1 converges to the optimal value of Problem (2.19) within a finite number of iterations. Intuitively, finite termination is guaranteed since the feasible set is finite and the outer-approximation process of Algorithm 1 never visits a point twice. We also remark that we need not solve a new binary optimization problem at each iteration of Algorithm 1 by integrating the entire algorithm within a single branch-and-bound tree, as proposed by [189], using lazy constraint callbacks.

Theorem 2. *Algorithm 1 terminates and returns an optimal solution to Problem* (2.19) *in a finite number of iterations.*

Noting that, from Theorem 1, $f(z) = -\frac{\boldsymbol{\mu}^\top \boldsymbol{\beta}^*(\boldsymbol{z})}{2}$ is convex in z, where $\boldsymbol{\beta}^*(\boldsymbol{z}) = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z}\boldsymbol{\mu}$, and that zero is always a feasible solution, we can conclude on termination and convergence of the outer-approximation cutting plane algorithm (described in Algorithm 1) by application of the classic result from [107].

2.6 An Efficient Heuristic Algorithm

In this section, we develop a heuristic algorithm for solving the MIO formulation defined by Equations (2.1)-(2.4) fast, primarily to obtain good starting points for the cutting plane algorithm (Algorithm (1)).

The following lemma provides an upper bound for Problem (2.1)-(2.4):

Lemma 5. *Denote by $\mathcal{Z}_\beta$ the feasible set defined by Equations* (2.2)-(2.4)*. Then we have*

$$\begin{aligned}
&\min_{\boldsymbol{\beta}\in\mathcal{Z}_\beta} \quad \sum_{n=1}^{N}\sum_{t=1}^{T}\left(y_n^t - \sum_{d=1}^{D} X_{n,d}^t \beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\sum_{d=1}^{D}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{(s,t)\in E}\sum_{d=1}^{D}\left(\beta_d^t - \beta_d^s\right)^2 \\
&\leq \min_{\boldsymbol{\beta}\in\mathcal{Z}_\beta} \quad \frac{1}{D}\sum_{n=1}^{N}\sum_{t=1}^{T}\sum_{d=1}^{D}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\sum_{d=1}^{D}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{t=1}^{T}\sum_{d=1}^{D} 2d^t\left(\beta_d^t\right)^2.
\end{aligned}$$

The proof, which we defer to Section 2.9, is based on the observation that the prediction error of the best multivariate model is less than or equal to the error of any univariate model. The above manipulations enable us to obtain a (possibly loose) upper bound to the original optimization problem, which however is additively separable in the optimization variables. The interpretation of the new optimization problem is as follows: we now fit separate univariate regressions per vertex per feature; we approximate the slow variation penalty with a new regularization term that depends on the degree of each vertex; we keep all sparsity and slow variation constraints.

We then proceed similarly to Section 2.2. We introduce binary variables $\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w}$ to capture the local sparsity, global sparsity, and sparsely varying support requirements. Importantly, we now require that the local sparsity requirement is *exactly* enforced, that is, $\sum_{d=1}^{D} z_d^t = K_{\mathrm{L}},\ \forall t \in [T]$; we denote by $\mathcal{Z}_=$ the corresponding binary feasible set. We replace every occurrence of β_d^t with $z_d^t\beta_d^t$. The resulting MIO formulation can then be written as:

$$\begin{aligned}
&\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}_=} \ \min_{\boldsymbol{\beta}} \quad \sum_{t=1}^{T}\sum_{d=1}^{D}\frac{1}{D}\sum_{n=1}^{N}\left(y_n^t - X_{n,d}^t z_d^t\beta_d^t\right)^2 + \lambda_\beta (z_d^t\beta_d^t)^2 + \lambda_\delta 2d^t (z_d^t\beta_d^t)^2 \\
&= \min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}_=} \ \min_{\boldsymbol{\beta}} \quad \sum_{t=1}^{T}\sum_{d=1}^{D}\left\{\left[\frac{1}{D}\sum_{n=1}^{N}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta(\beta_d^t)^2 + \lambda_\delta 2d^t(\beta_d^t)^2\right] z_d^t + \frac{1}{D}\sum_{n=1}^{N}(y_n^t)^2(1-z_d^t)\right\} \\
&= \underbrace{\frac{1}{D}\sum_{t=1}^{T}\sum_{n=1}^{N}(y_n^t)^2(D-K_L)}_{:=L_0} + \min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}_=} \quad \sum_{t=1}^{T}\sum_{d=1}^{D}\underbrace{\min_{\beta_d^t}\left[\frac{1}{D}\sum_{n=1}^{N}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta(\beta_d^t)^2 + \lambda_\delta 2d^t(\beta_d^t)^2\right]}_{:=L_d^t} z_d^t \\
&:= L_0 + \min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}_=} \sum_{t=1}^{T}\sum_{d=1}^{D} L_d^t z_d^t. \qquad (2.22)
\end{aligned}$$

As shown in Equation (2.22), due to separability, we can solve the inner problem in closed form and obtain an integer linear optimization problem in the binary variables. The polyhedral feasible set of the corresponding linear relaxation, which we denote by $\tilde{\mathcal{Z}}_=$, can unfortunately be shown to not be integral. Nevertheless, we empirically

show in Section 2.10.5 that the solution to the linear relaxation is, in fact, integral or near-integral, for a variety of realistic, non-pathological problems. Therefore, we proceed by solving the linear relaxation of Problem (2.22). In case the solution $\boldsymbol{z}^{\mathrm{H}}$ to the linear relaxation is not binary feasible, we round up all non-integral entries. Finally, to ensure feasibility in terms of the local and global sparsity, as well as the sparsely varying constraints, we iteratively remove features from the global support until the resulting solution is indeed feasible.

The final heuristic algorithm is given in Algorithm 2. Proposition 2, proved in Section 2.9, asserts that Algorithm 2 gives a feasible solution for Problem (2.1)-(2.4) in polynomial time:

Proposition 2. *Algorithm 2 terminates and provides a feasible solution to Problem* (2.1)-(2.4) *in time* $\tilde{O}\left(NTD + (TD)^{2+1/6} + T^2K_L^2(T + K_L)\right)$.

2.7 Experiments on Synthetic Data

In this section, we evaluate the proposed SSVR framework using synthetic data. We present a high-level description of our experimental methodology, as well as a small set of aggregated and selected computational results, and defer the details to Section 2.10.2.

2.7.1 Methodology and Implementation Details

We compare proposed algorithms (Algorithm 1 referred to as `svar_cutplane` and Algorithm 2 referred to as `svar_heuristic`) with the cutting plane algorithm of [35] that solves the standard sparse regression formulation shown in Problem (2.6) (referred to as `sparse_regression`), as well as a suite of 4 variants of the sum-of-norms regularization framework of [178] shown in Problem (2.5) with and without lasso regularization (referred to as `sum_of_norms_l1`, `sum_of_norms_l1_lasso`, `sum_of_norms_l2`, `sum_of_norms_l2_lasso`). The implementation details of all methods, e.g., programming language and software used, and the computing environment in which we run

Algorithm 2: Heuristic Algorithm

Input: Data $(\boldsymbol{X}^t, \boldsymbol{y}^t)_{t=1}^T$, similarity graph G, sparsity parameters $(K_{\mathrm{L}}, K_{\mathrm{G}}, K_{\mathrm{C}})$, regularization parameters $(\lambda_\beta, \lambda_\delta)$.

Output: Learned coefficients $\boldsymbol{\beta}^H$.

▷ `Compute loss for each vertex-feature pair:`

for $t \in [T],\ d \in [D]$ **do**

$L_d^t \leftarrow \min_{\beta_d^t} \frac{1}{D} \sum_{n=1}^N \left(y_n^t - X_{n,d}^t \beta_d^t \right)^2 + \lambda_\beta (\beta_d^t)^2 + \lambda_\delta 2 d^t (\beta_d^t)^2$

end for

▷ `Solve linear relaxation of Problem` (2.22)`:`

$(\boldsymbol{z}^H, \boldsymbol{s}^H, \boldsymbol{w}^H) \leftarrow \min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w} \in \tilde{\mathcal{Z}}_=} L_d^t z_d^t$

▷ `Ensure integrality:`

if $\boldsymbol{z}^H \notin \{0,1\}^{TD}$ **then**

▷ `Find non-integral entries in` $\boldsymbol{z}^H$`.`

$\mathcal{I} \leftarrow \{(t,d) \in [T] \times [D] :\ 0 < (\boldsymbol{z}^H)_d^t < 1\}$

$(\boldsymbol{z}^H)_d^t \leftarrow 1,\ \forall (t,d) \in \mathcal{I}$

▷ `Accordingly update` $\boldsymbol{s}^H, \boldsymbol{w}^H$ `(as per Section 2.2).`

$(\boldsymbol{s}^H, \boldsymbol{w}^H) \leftarrow f(\boldsymbol{z}^H)$

end if

▷ `Ensure feasibility:`

while $(\boldsymbol{z}^H, \boldsymbol{s}^H, \boldsymbol{w}^H) \notin \mathcal{Z}_=$ **do**

▷ `Find global support.`

$\mathcal{S} \leftarrow \{d \in [D] :\ \sum_{t=1}^T (\boldsymbol{z}^H)_d^t > 0\}$

▷ `Find feature` $d \in \mathcal{S}$ `with largest average loss across all vertices.`

$d_0 \leftarrow \mathrm{argmax}_{d \in \mathcal{S}} \frac{1}{T} \sum_{t=1}^T L_d^t$

$(\boldsymbol{z}^H)_d^t \leftarrow 0,\ \forall t \in [T]$

▷ `Accordingly update` $\boldsymbol{s}^H, \boldsymbol{w}^H$ `(as per Section 2.2).`

$(\boldsymbol{s}^H, \boldsymbol{w}^H) \leftarrow f(\boldsymbol{z}^H)$

end while

▷ `Estimate coefficients using Theorem 1:`

$\boldsymbol{\beta}^H \leftarrow \boldsymbol{\beta}^*(\boldsymbol{z}^H)$

return $\boldsymbol{\beta}^\star$

our experiments are described in Section 2.10.1. For each method, we tune the regularization and slowly varying hyperparameters (λ_β and λ_δ) using holdout validation and exhaustive grid search over the same range of values. In the results we report in Sections 2.7.3 and 2.7.2, we assume that sparsity-related hyperparameters (K_{L}, K_{G}, and K_{C}) are known; we investigate the impact of imperfect sparsity-related hyperparameters in Section 2.7.4.

For our synthetic data experiments, we generate a number of synthetic datasets according to the following methodology. We create a matrix of sparse and slowly varying regression coefficients $\boldsymbol{\beta} \in \{-1, 0, 1\}^{T \times D}$ over a (known) Erdos-Renyi similarity graph G. We then create a random data matrix $\boldsymbol{X} \in \mathbb{R}^{N \times T \times D}$ with Toeplitz correlation structure across features, and use the ground truth coefficients $\boldsymbol{\beta}$ to generate noisy responses $\boldsymbol{Y} \in \mathbb{R}^{N \times T}$. Our data generation methodology involves a number of problem parameters: the number of data points (N), the number of features (D), the number of vertices in the similarity graph (T), the density of the similarity graph d_G, the level of variation of the regression coefficients between adjacent vertices (σ_V), the sparsity parameters (K_L, K_G, K_C), the correlation between features (ρ_d), and the signal-to-noise ratio (ξ) in the generated data. We generate different datasets by varying the above parameters. The details of our data generation methodology are provided in Section 2.10.2.

We use various evaluation metrics that aim to assess different aspects of the regression problem at hand: estimation accuracy, predictive power, computational efficiency, and efficiency of the cutting plane method. Specifically, we consider: mean absolute error in the estimated coefficients (MAE), support recovery accuracy (ACC), and support recovery false alarm rate (FA) to assess each method's estimation accuracy; out-of-sample R^2 statistic (Test R^2) to assess each method's predictive power; computational time (Time) to assess each method's computational efficiency; optimality gap, number of cuts (Cut Count), and average cut time (ACT) to compare the performance of the proposed cutting plane method (Algorithm 1) against the cutting plane method of [35]. We outline the above in more detail in Table 2.5 in Section 2.10.2.

2.7.2 Aggregated Sensitivity Analysis with Known Sparsity Parameters

In this section, we report aggregated results from our sensitivity analyses with respect to all problem parameters; we obtain such results using the following methodology.

We consider one problem parameter at a time, and evaluate the performance of all methods as a function of such parameter, while keeping all other parameters fixed. In total, we setup a series of 43 experiments, each of which corresponds to a fixed setting of the problem parameters $(N, T, D, K_{\mathrm{L}}, K_{\mathrm{G}}, K_{\mathrm{C}}, \sigma_v, d_G, \rho_d, \xi)$.

For each experiment (or, equivalently, problem parameter setting), we independently generate 10 datasets. For each method, we impose a time limit of 900 seconds and compute the mean and standard deviation of each evaluation metric across those 10 datasets; if no solution is returned when the solver terminates, we return the all-zeros solution. We note that the solver might not terminate exactly at the time limit, but will instead stop after performing the required computations of the attributes associated with the terminated optimization [124]. Moreover, if the solution time of a method exceeded 1 hour in preliminary experiments, we did not include this method in our reported experiments.

We rank the methods according to their performance. We report the mean and standard deviation of the rank of each method across all experiments. Table 2.1 presents the aggregated results (obtained over 430 datasets); the key takeaways are the following:

Table 2.1: Aggregated sensitivity analysis: mean and std. of each method's ranking across all experiments.

Algorithm	MAE	ACC	FA	Test R^2	Time	Gap	Cut Count	ACT
`svar_cutplane`	**1.33 (0.52)**	3.35 (1.04)	**1.79 (0.64)**	1.93 (0.46)	3.05 (0.62)	**1.49 (0.51)**	**1.0 (0.0)**	**1.4 (0.49)**
`svar_heuristic`	4.49 (0.8)	4.81 (0.76)	3.33 (0.94)	4.77 (0.78)	**1.0 (0.0)**	-	-	-
`sparse_regression`	1.95 (0.84)	2.95 (0.92)	1.81 (1.01)	3.63 (0.9)	2.19 (0.59)	1.51 (0.51)	2.0 (0.0)	1.6 (0.49)
`sum_of_norms_l1`	3.98 (0.8)	**1.37 (0.98)**	5.02 (0.71)	**1.37 (0.98)**	4.6 (0.73)	-	-	-
`sum_of_norms_l1_lasso`	3.44 (1.16)	2.79 (1.34)	3.84 (1.23)	3.58 (1.05)	4.35 (0.81)	-	-	-
`sum_of_norms_l2`	7.0 (0.0)	6.88 (0.54)	6.74 (1.07)	6.84 (0.75)	6.86 (0.52)	-	-	-
`sum_of_norms_l2_lasso`	6.53 (0.88)	6.56 (0.88)	6.19 (1.68)	6.6 (0.73)	6.67 (0.68)	-	-	-

- *Estimation Accuracy:* `svar_cutplane` achieves the best average MAE and FA, whereas `sum_of_norms_l1` achieves the best ACC (and one of the worst FA) as

it keeps most features in its set.

- *Predictive power:* `sum_of_norms_l1` leads Test R^2, closely followed by `svar_cutplane` (which is not statistically significantly worse). The slight increase in the accuracy of `sum_of_norms_l1` compared with `svar_cutplane` comes at a cost of a significantly more dense model, making it difficult to interpret. All other methods present a significant gap in accuracy.

- *Computational Efficiency:* `svar_heuristic` is the clear winner, having ranked first and by a large margin in all problem settings we considered. The cutting plane-based methods follow, with `sparse_regression` being slightly faster than `svar_cutplane`, as the number of estimated parameters is D for `sparse_regression` as opposed to TD for `svar_cutplane`, which fits T related sparse regression models. This is then followed by the L1 regularization methods `sum_of_norms_l1` and `sum_of_norms_l1_lasso`, with the L2 regularization methods `sum_of_norms_l2` and `sum_of_norms_l2_lasso` exhibiting the worst scaling behavior.

- *Evaluation of the Cutting Plane Method:* `svar_cutplane` performs better than `sparse_regression` on the cutting plane method, being better on average at proving optimality (Gap), always generates fewer cuts, and faster on average at generating cuts.

Overall, the proposed `svar_cutplane` achieves state-of-the-art out-of-sample performance and coefficient recovery faster than competing methodologies, and does so with an accurate sparse model that aids interpretability. Moreover, the proposed `svar_heuristic` achieves similar performance with the competing methodologies, but is able to do so significantly faster. In the following Section 2.7.3, we further illustrate this conclusion, focusing on experiments varying a key parameter, T.

2.7.3 Sensitivity Analysis for Varying Number of Vertices in the Similarity Graph

In this section, we investigate the performance and scalability of all methods as a function of number of vertices T, the key parameter that motivates the proposed framework and, as we show, drives the complexity of all methods. We set remaining problem parameters to certain defaults (see Section 2.10.2). We present the results in Figure 2-3.

Estimation Accuracy. The results match the conclusion in Section 2.7.2. Figure 2-3(a) suggests that `svar_cutplane` achieves the smallest MAE for almost all values of T under consideration, and the quality of the estimated coefficients does not deteriorate with T. In terms of support recovery, `sum_of_norms_l1` and `sum_of_norms_l2` achieve the highest ACC, followed by `sum_of_norms_l1_lasso` and `sum_of_norms_l2_lasso`, and outperforming `svar_cutplane` and `sparse_regression` (Figure 2-3(b)). This, however, comes at the expense of an extremely high FA, especially for `sum_of_norms_l1` and `sum_of_norms_l2`; in contrast the FA for `svar_cutplane` and `sparse_regression` is near zero (Figure 2-3(c)). Finally, we note that, although `svar_heuristic` is in general weaker, it is able to accurately estimate the top $\frac{1}{3}$ to $\frac{1}{2}$ of the relevant features and, as we show next the gap in performance is partly compensated by its computational efficiency.

Predictive Power. In out-of-sample predictive power, Figure 2-3(d) shows that non-regularized methods `sum_of_norms_l1` and `sum_of_norms_l2` achieve the highest Test R^2; `svar_cutplane` comes close second, followed by `sum_of_norms_l2_lasso` and `sum_of_norms_l1_lasso`, and then by `sparse_regression` and `svar_heuristic`. We remark however, that the high Test R^2 achieved by `sum_of_norms_l1` and `sum_of_norms_l2` comes at the expense of interpretability, as the learned models are fully dense, assigning nonzero coefficients to nearly all features.

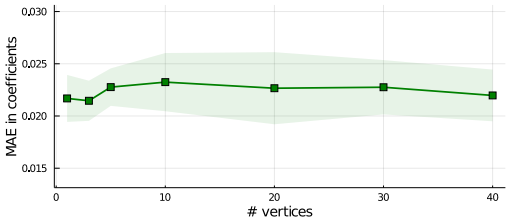
Computational Efficiency. As shown in Figure 2-3(e), `svar_heuristic` is extremely fast, converging in $< 1s$ in problems with $T \leq 10$ and $< 5s$ in larger problems. The cutting plane-based methods follow, with `sparse_regression` being slightly faster than `svar_cutplane`, as the number of estimated parameters is D for `sparse_regression` as opposed to TD for `svar_cutplane`, which fits T related sparse regression models. `sum_of_norms_l1` and `sum_of_norms_l1_lasso` scale to problems with $T \leq 20$ within one hour while `sum_of_norms_l2` and `sum_of_norms_l2_lasso` scale more unfavorably, exceeding the one hour limit for problems with $T > 5$.

Evaluation of the Cutting Plane Method. Finally, we compare the proposed cutting plane method (Algorithm 1) with [35]. Again, we see the conclusion in Section 2.7.2 holds. `svar_cutplane` consistently finds near-optimal solutions, with an optimality gap of less than 0.05, whereas `sparse_regression`'s optimality gap increases with T (Figure 2-3(f)). Figure 2-3(g) shows that, by exploiting the problem structure, `svar_cutplane` generates vastly fewer cuts than `sparse_regression`. Finally, in Figure 2-3(h), observe that the average cut generation time increases with T for both methods: for `svar_cutplane`, the increase is quadratic, in agreement with Lemma 4; for `sparse_regression`, the increase is linear in the total number of data points $N' = NT$ (in agreement with the analysis of [35]).

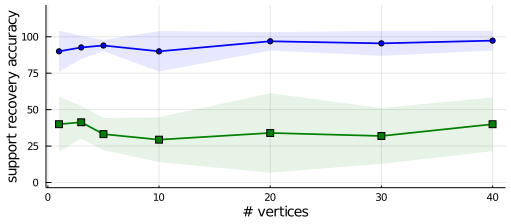
2.7.4 Impact of Unknown Sparsity Parameters

In our synthetic experiments so far, we have assumed that the sparsity-related hyperparameters (K_{L}, K_{G}, K_{C}) are known. In this section, we drop this assumption and explore how accurately `svar_cutplane`, `svar_heuristic`, and `sparse_regression`, i.e., the methods that exactly impose sparsity, can recover the sparsity-related hyperparameters using simple holdout validation and exhaustive grid search (over a grid of 27 parameter combinations).

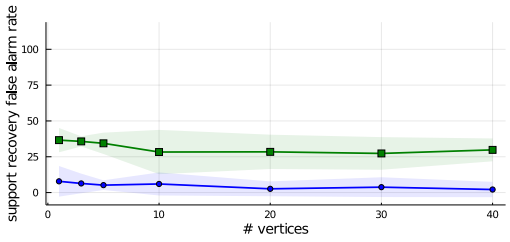
We use default problem parameters (see Section 2.10.2), except increasing the number of data points N by 50% to account for unknown sparsity-related parameters. We split N data points into training and validation sets. For each method, we compare



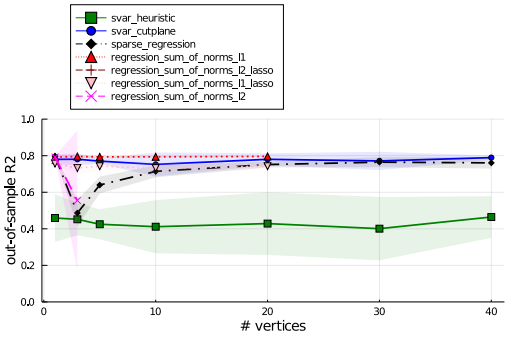
(a) Mean absolute error in estimated coefficients.



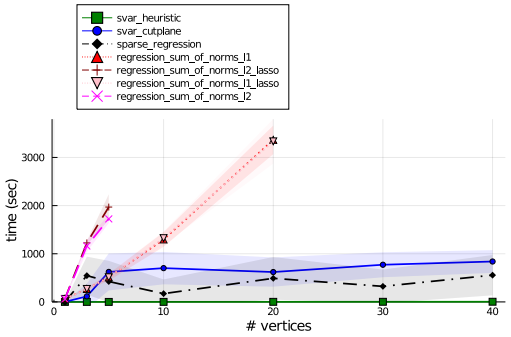
(b) Support recovery accuracy.



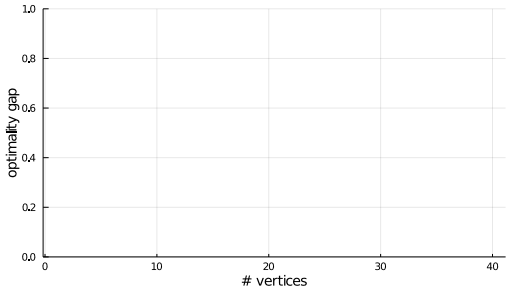
(c) Support recovery false alarm rate.



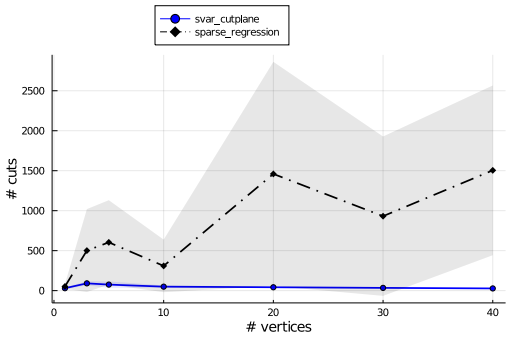
(d) Out-of-sample R^2.



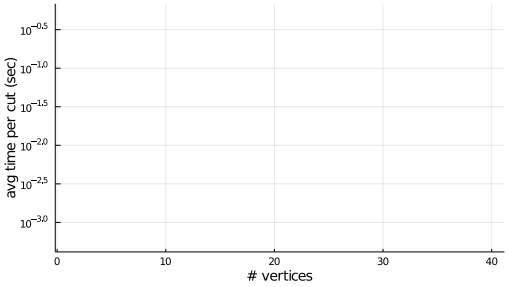
(e) Computational time (in seconds).



(f) Optimality gap.



(g) Number of cuts.



(h) Average time per cut.

Figure 2-3: Sensitivity analysis for varying number of vertices T in the similarity graph G.

sparsity parameters in the final model ($\hat{K}_{\mathrm{L}}$, $\hat{K}_{\mathrm{G}}$, $\hat{K}_{\mathrm{C}}$) with ground truth sparsity parameters (K_{L}, K_{G}, K_{C}). Note the sparsity parameters selected in validation may differ from the sparsity parameters in the reported final model, due to $\leq$ constraints in Problem (2.1)-(2.4) for `svar_cutplane` and `svar_heuristic`, and in Problem (2.6) for `sparse_regression`. The results are in Table 2.2, where we also report the final model's ACC and FA.

Table 2.2: Unknown sparsity hyperparameters: hyperparam. estimation accuracy and support recovery performance.

Algorithm	$\lvert\hat{K}_L - K_L\rvert$	$\lvert\hat{K}_G - K_G\rvert$	$\lvert\hat{K}_C - K_C\rvert$	**ACC**	**FA**
`svar_cutplane`	**3.0 (2.58)**	3.1 (2.88)	**4.9 (6.4)**	90.6 (13.0)	**33.36 (25.47)**
`svar_heuristic`	4.5 (0.85)	4.1 (0.88)	6.4 (3.34)	53.0 (27.97)	51.95 (10.73)
`sparse_regression`	3.5 (2.42)	**2.6 (0.52)**	25.8 (2.74)	96.2 (1.48)	36.79 (22.74)
`sum_of_norms_l1`	195.0 (0.0)	192.9 (0.32)	25.8 (2.74)	**100.0 (0.0)**	97.5 (0.0)
`sum_of_norms_l1_lasso`	50.7 (56.52)	69.9 (67.76)	487.8 (393.28)	85.6 (15.57)	63.99 (34.01)

Table 2.2 suggests that `svar_cutplane` is on average the best at recovering the true local sparsity parameter K_{L}. `sparse_regression` ranks second in terms of K_{L} and first in terms of recovering the true global sparsity parameter K_{G}; as it cannot capture different local and global sparsity levels, it tends to select denser models, where all features relevant in at least one vertex are included. Both `svar_cutplane` and, perhaps surprisingly, `svar_heuristic` are strong in accurately capturing the sparsely varying support of the underlying model K_{C}. Finally, ACC and FA results are consistent with those presented in Sections 2.7.3 and 2.7.2, where the sparsity parameters were known.

2.8 Experiments on Real-World Data

In this section, we study the performance of the proposed SSVR framework on publicly available real-world data. We consider two datasets with temporally varying structure and three datasets with spatially varying structure; first, we present aggregated computational results across a variety of metrics (Section 2.8.1; then we delve deeper into each dataset and discuss the models learned by the proposed framework (Sections 2.8.2 and 2.8.3). In Section 2.10.3, we provide additional information on the datasets

and the preprocessing we to each of them (Section 2.10.3), as well as more detailed computational results (Section 2.10.3).

2.8.1 Aggregated Results

We obtain the aggregated results we report in this section as follows.

We randomly split each dataset 10 times into training (60%), validation (20%), and test (20%) sets (respecting the temporal structure if such exists). For each dataset and each metric, we compute the mean and standard deviation of each method across those 10 splits. We rank the methods according to their performance. We report the mean and standard deviation of the rank of each method across all experiments. Table 2.3 presents the aggregated results (obtained over 50 datasets).

In agreement with our conclusions from Section 2.7, `sum_of_norms_l1`, closely followed by `svar_cutplane` are the winners in terms of their *predictive power*; we also emphasize the surprisingly good performance of `svar_heuristic`. As in real-world problems there is no way to assess estimation accuracy, we instead focus on model *interpretability*; we report each method's estimated local sparsity ($\hat{K}_{\mathrm{L}}$), where `svar_cutplane` and `svar_heuristic` produce, in general, simpler and hence more interpretable models, as well as each method's estimated global sparsity ($\hat{K}_{\mathrm{G}}$), where `svar_heuristic` uses the least number of features; it is unclear whether a smaller or bigger number of changes in support $\hat{K}_{\mathrm{C}}$ is preferred (`sum_of_norms_l1`ranks first because it always has all features in its support). Finally, similar to the synthetic experiments, `svar_heuristic` is the clear winner on *computational time*, and `svar_cutplane` significantly outperforms `sparse_regression` on proving optimality and on generating fewer and faster cuts, i.e., on the *evaluation of the cutting plane method*.

2.8.2 Datasets with Temporally Varying Structure

In this section, we delve deeper into the temporal datasets and the corresponding learned models.

Table 2.3: Aggregated results for real-world data: mean and std. of each method's ranking across all experiments.

Algorithm	Test R^2	Local Sparsity	Global Sparsity	Changes in Support	Time	Gap	ACT	Cut Count
svar_cutplane	2.4 (0.89)	**2.2 (0.84)**	2.2 (0.45)	3.4 (0.55)	3.2 (1.79)	**1.4 (0.55)**	**1.4 (0.55)**	**1.2 (0.45)**
svar_heuristic	3.4 (0.89)	2.2 (1.1)	**1.6 (0.89)**	2.8 (0.84)	**1.2 (0.45)**	-	-	-
sparse_regression	5.4 (1.79)	3.4 (1.52)	-	-	2.8 (0.84)	1.6 (0.55)	1.6 (0.55)	1.8 (0.45)
sum_of_norms_l1	**1.8 (1.3)**	5.6 (0.55)	4.4 (0.55)	1.0 (0.0)	3.8 (1.1)	-	-	-
sum_of_norms_l1_lasso	5.6 (0.84)	2.2 (1.64)	2.2 (1.1)	4.8 (0.84)	4.0 (1.0)	-	-	-
sum_of_norms_l2	3.4 (2.41)	6.0 (0.0)	4.8 (0.45)	2.6 (2.19)	6.0 (0.0)	-	-	-
sum_of_norms_l2_lasso	5.6 (0.55)	4.8 (1.64)	4.4 (0.55)	5.4 (0.55)	6.6 (0.55)	-	-	-

Appliances Energy Prediction: Hourly. In this experiment, we focus on a real-world case study concerned with appliances energy prediction [70]. Each observation in the dataset is a vector of measurements (temperature and humidity in various rooms, weather conditions, etc.) in a low energy building, and the goal is to predict the energy consumption of the building's appliances. After preprocessing the dataset (see Section 2.10.3 for details), we get $N = 822$ data points per vertex, $T = 24$ vertices (each corresponding to an hour of the day), and $D = 26$ features. To capture the temporal structure of the problem, the similarity graph is a chain (see Figure 2-1 (left)).

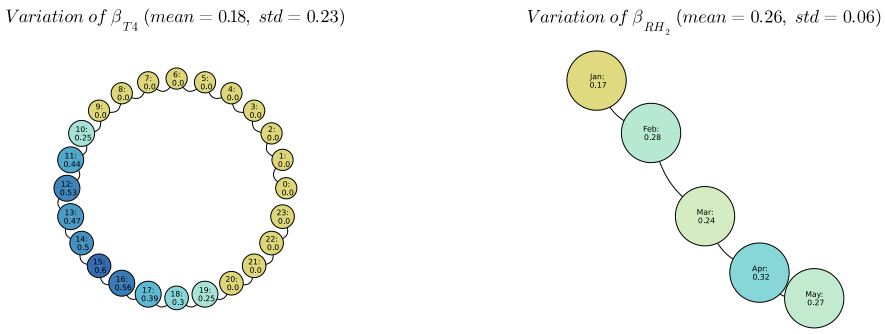
Our open-source implementation outputs the final model as a graph, with structure matching that of the underlying similarity graph. Each vertex shows the learned regression coefficient for any selected feature at the corresponding vertex of the similarity graph; vertices in yellow (resp. blue) correspond to coefficients below (resp. above) the mean across all vertices.

Figure 2-4(a) presents the variation of the regression coefficient with the highest mean absolute magnitude across all vertices in the best `svar_cutplane` model. The corresponding feature, T4, corresponds to the temperature in the office room. $\boldsymbol{\beta}_{\mathrm{T4}}$ is zero between 8pm and 9am, when the office room is likely empty, slowly increases during the day, peaks in the afternoon, and then slowly decreases in the evening. The slowly and sparsely varying structure of the learned model is clear.

Appliances Energy Prediction: Monthly. In this experiment, we consider the same appliances energy dataset. However, instead of assigning a vertex to each hour of the day, we now assign a vertex to each month. We get $N = 2,922$ data points per

vertex, $T = 5$ vertices (the data covers from January and May), and $D = 26$ features. The similarity graph is again a chain.

Figure 2-4(b) presents the variation of the regression coefficient with the highest mean absolute magnitude across all vertices in the best `svar_cutplane` model. In this case, the corresponding feature, RH2, corresponds to the living room area humidity. $\boldsymbol{\beta}_{\text{RH2}}$ again varies slowly across months.



(a) Appliances Energy Prediction: Hourly. (b) Appliances Energy Prediction: Monthly.

Figure 2-4: Variation of most important feature across vertices on datasets with temporally varying structure.

2.8.3 Datasets with Spatially Varying Structure

In this section, we discuss the details of the spatial datasets and the corresponding learned models.

Housing Price Prediction. In this experiment, we explore the application of our framework to housing price prediction in Ames, Iowa [87]. The dataset consists of a number of features involved in assessing home values, and the goal is to predict the selling price of the home. After preprocessing the dataset, we get $N = 822$ data points per vertex, $T = 7$ vertices (each corresponding to a cluster of neighborhoods in Ames, Iowa; see Figure 2-5(a) for a visualization), and $D = 199$ features. The similarity graph connects adjacent neighborhood clusters and consists of $E = 8$ edges.

Figure 2-5(b) presents the variation of the regression coefficient with the highest mean absolute magnitude across all vertices in the best `svar_cutplane` model. In

this case, the corresponding feature, GrLivArea, corresponds to the above ground living area. $\boldsymbol{\beta}_{\text{GrLivArea}}$ peaks in the NW and NE neighborhood clusters, and its value decreases in the southernmost clusters (W, S, E).

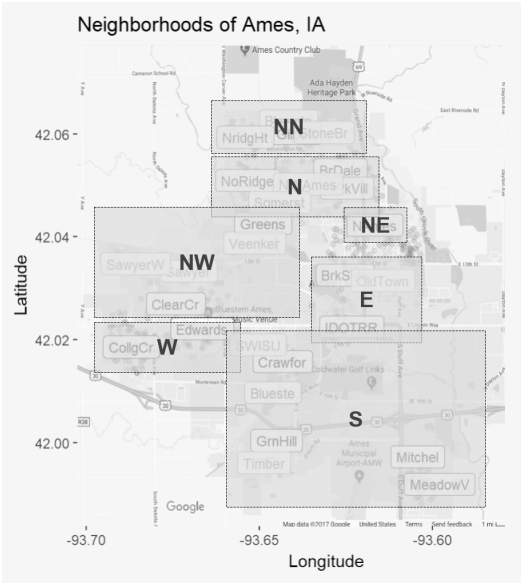
Air Quality. In this experiment, we consider air quality prediction in 12 air quality monitoring sites in Beijing [237]. The original dataset consists of weather (temperature, pressure, dew point temperature, precipitation, wind speed, wind direction) and time-related features, and the goal is to predict PM2.5 concentration - an air pollutant that is a health concern at high levels. After preprocessing the dataset, we get $N = 35,064$ data points per vertex, $T = 12$ vertices (each corresponding to an air quality monitoring site), and $D = 25$ features. The similarity graph connects adjacent sites and consists of $E = 14$ edges and 4 connected components.

Figure 2-5(c) presents the variation of the regression coefficient with the highest mean absolute magnitude across all vertices in the best `svar_cutplane` model. In this case, the corresponding feature, DEWP, corresponds to the dew point temperature. The benefits of the proposed SSVR framework are again clear: the range of values for $\boldsymbol{\beta}_{\text{DEWP}}$ is between 0.61 and 0.72; however, across all connected components, the maximum coefficient variation never exceeds 0.06.

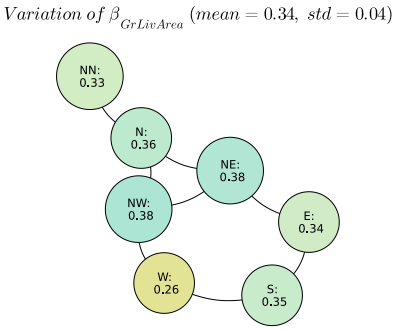
Meteorology. In this experiment, we consider the task of weather prediction in 30 US and Canadian Cities, as well as 6 Israeli cities. The original dataset contains hourly measurements of weather attributes (temperature, humidity, air pressure, wind direction, and wind speed), and the goal is to predict the temperature half a day in advance. After preprocessing the dataset, we get $N = 45,231$ data points per vertex, $T = 36$ vertices (each corresponding to a city), and $D = 50$ features. The similarity graph connects adjacent sites and consists of $E = 110$ edges and 2 connected components.

Figure 2-5(d) presents the variation of the regression coefficient with the highest mean absolute magnitude across all vertices in the best `svar_cutplane` model. In this case, the corresponding feature, T1, corresponds, perhaps unsurprisingly, to the

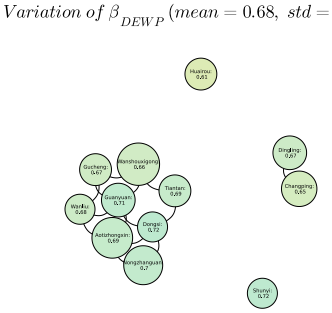
current temperature. The visualization of the learned model clearly shows how $\boldsymbol{\beta}_{\mathrm{T1}}$ varies slowly across the similarity graph.



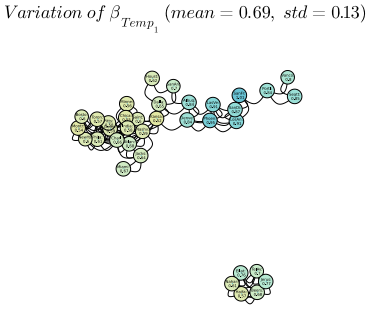
(a) Neighborhood clusters in Ames, IA.



(b) Housing Price Prediction.



(c) Air Quality.



(d) Meteorology.

Figure 2-5: Variation of most important feature across vertices on datasets with spatially varying structure.

2.9 Technical Proofs

Proof of Lemma 1. Note that by rearranging the variables, we can rewrite the optimization problem as:

$$\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}\in\mathcal{Z}} \min_{\boldsymbol{\beta}} \quad c(\boldsymbol{z},\boldsymbol{\beta}) := \boldsymbol{\beta}^\top(\boldsymbol{Z}(\boldsymbol{M}+\lambda_\beta \boldsymbol{I})\boldsymbol{Z})\boldsymbol{\beta} - 2\boldsymbol{\mu}^\top \boldsymbol{Z}\boldsymbol{\beta} + \sum_{t=1}^{T} \|\boldsymbol{y}^t\|_2^2$$

where $\boldsymbol{\beta} = (\boldsymbol{\beta}^1, \cdots, \boldsymbol{\beta}^T)$, $\boldsymbol{Z} = \text{Diag}(\boldsymbol{z}^1, \cdots, \boldsymbol{z}^T)$, and $\mathcal{Z}$ is the polyhedral feasible set as defined by the binary constraints on $\boldsymbol{z}, \boldsymbol{s}, \boldsymbol{w}$ and (2.11)–(2.16), and $\boldsymbol{M}$ can be defined as:

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{X}^1 \\ \boldsymbol{X}^2 \\ \vdots \\ \boldsymbol{X}^T \end{pmatrix} \begin{pmatrix} \boldsymbol{X}^1 & \boldsymbol{X}^2 & \dots & \boldsymbol{X}^T \end{pmatrix} + \begin{pmatrix} \mathbb{1}_{\exists t,(1,t)\in E} - \mathbb{1}_{\exists s,(s,1)\in E} \\ \mathbb{1}_{\exists t,(2,t)\in E} - \mathbb{1}_{\exists s,(s,2)\in E} \\ \vdots \\ \mathbb{1}_{\exists t,(T,t)\in E} - \mathbb{1}_{\exists s,(s,T)\in E} \end{pmatrix} \times \left(\begin{pmatrix} \mathbb{1}_{\exists t,(1,t)\in E} - \mathbb{1}_{\exists s,(s,1)\in E} \\ \mathbb{1}_{\exists t,(2,t)\in E} - \mathbb{1}_{\exists s,(s,2)\in E} \\ \vdots \\ \mathbb{1}_{\exists t,(T,t)\in E} - \mathbb{1}_{\exists s,(s,T)\in E} \end{pmatrix} \right)^T$$

It is clear that the both matrices in the right expression are positive semidefinite, and thus $\boldsymbol{M}$ is positive semidefinite. We then reach the final form by dividing the objective by 2 and noting $\sum_{t=1}^{T} \|\boldsymbol{y}^t\|_2^2$ is a constant within the optimization problem and thus can be removed. □

Proof of Lemma 2. We first note that our minimization problem is equivalent to the sum-of-norms original problem. Combining this with the fact that $\boldsymbol{M}$ is positive semi-definite, to solve the inner problem we only need to derive the first order condition, which is:

$$\frac{\partial c(\boldsymbol{z}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = (\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})\boldsymbol{\beta} - \boldsymbol{Z}\boldsymbol{\mu} = \boldsymbol{0}.$$

$\boldsymbol{Z}$ is a rank $\leq TK_{\text{L}} < TD$ matrix, so $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})$ is rank-deficient, and thus to solve this first-order condition, we can utilize the Moore-Penrose pseudo-inverse to write:

$$\boldsymbol{\beta}^*(\boldsymbol{z}) = (\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z}\boldsymbol{\mu}.$$

The second assertion follows from substituting the first order equality $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})\boldsymbol{\beta}^*(\boldsymbol{z}) = \boldsymbol{Z}\boldsymbol{\mu}$ into the objective expression. We have:

$$\frac{1}{2}\boldsymbol{\beta}^*(\boldsymbol{z})^\top (\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})\boldsymbol{\beta}^*(\boldsymbol{z}) - \boldsymbol{\mu}^\top \boldsymbol{Z}\boldsymbol{\beta}^*(\boldsymbol{z}) = -\frac{1}{2}\boldsymbol{\beta}^*(\boldsymbol{z})^\top \boldsymbol{Z}\boldsymbol{\mu} = -\frac{1}{2}\boldsymbol{\beta}^*(\boldsymbol{z})^\top \boldsymbol{\mu}$$

As we note $\boldsymbol{Z}\boldsymbol{\beta}^*(\boldsymbol{z}) = \boldsymbol{\beta}^*(\boldsymbol{z})$ by definition. □

Proof of Proposition 1. We prove this in two steps. First, we establish the following relation for the pseudoinverse:

Lemma 6. $(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1} - \lambda_\beta(\boldsymbol{I} - \boldsymbol{Z})$.

Proof of Lemma 6 We verify that the expression on the right satisfies the definition

of a Moore-Penrose pseudoinverse for $\boldsymbol{A} := \boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z}$. The Moore-Penrose pseudoinverse $\boldsymbol{A}^\dagger$ is the unique matrix that satisfies the four conditions: 1. $\boldsymbol{A}^\dagger \boldsymbol{A} \boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger$, 2. $\boldsymbol{A}\boldsymbol{A}^\dagger \boldsymbol{A} = \boldsymbol{A}$, 3. $(\boldsymbol{A}\boldsymbol{A}^\dagger)^* = \boldsymbol{A}\boldsymbol{A}^\dagger$, 4. $(\boldsymbol{A}^\dagger \boldsymbol{A})^* = \boldsymbol{A}^\dagger \boldsymbol{A}$, where $*$ is the Hermitian operator with $\boldsymbol{A}^*_{ij} = \overline{\boldsymbol{A}_{ji}}$.

The assertions follow immediately if we have $\boldsymbol{A}^\dagger \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\dagger = \boldsymbol{Z}$, which we next prove:

$$\begin{aligned}
\boldsymbol{A}^\dagger \boldsymbol{A} &= [(\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1} - \lambda_\beta(\boldsymbol{I} - \boldsymbol{Z})]\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z} \\
&= (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1}(\boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z} + \lambda_\beta \boldsymbol{Z}) \\
&= (\boldsymbol{I} - (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1}\lambda_\beta(\boldsymbol{I} - \boldsymbol{Z})) \\
&= (\boldsymbol{I} - \left(\frac{1}{\lambda_\beta}(\boldsymbol{I} - \boldsymbol{Z}(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{M}\boldsymbol{Z})\right)\lambda_\beta(\boldsymbol{I} - \boldsymbol{Z})) \\
&= \boldsymbol{Z},
\end{aligned}$$

where on the second last line we utilized the binomial inverse theorem [134]. Here, we have $\boldsymbol{Z}^2 = \boldsymbol{Z}$ as $\boldsymbol{Z}$ is a binary diagonal matrix. The case for $\boldsymbol{A}\boldsymbol{A}^\dagger$ is identical. □

then we note the following equivalence:

Lemma 7. $(\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z}$.

Proof of Lemma 7 By the binomial inverse theorem [134], we have:

$$(\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{Z} = \frac{1}{\lambda_\beta}(\boldsymbol{I} - \boldsymbol{Z}(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{M}\boldsymbol{Z})\boldsymbol{Z} = \frac{1}{\lambda_\beta}(\boldsymbol{Z} - \boldsymbol{Z}(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{M}\boldsymbol{Z}).$$

Similarly, we have:

$$(\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z} = \frac{1}{\lambda_\beta}(\boldsymbol{I} - \boldsymbol{Z}(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{M})\boldsymbol{Z} = \frac{1}{\lambda_\beta}(\boldsymbol{Z} - \boldsymbol{Z}(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1}\boldsymbol{M}\boldsymbol{Z}).$$

This proves the statement required. □

We now prove the final desired statement, utilizing Lemmata 6 and 7:

$$(\boldsymbol{Z}(\boldsymbol{M} + \lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z} = ((\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M}\boldsymbol{Z})^{-1} - \lambda_\beta(\boldsymbol{I} - \boldsymbol{Z}))\boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z}.$$

□

Proof of Theorem 1. The equivalence of the two optimization problems follows immediately from Lemma 2 and Proposition 1. We proceed to prove that $f(\boldsymbol{z}) = -\frac{\boldsymbol{\mu}^\top \boldsymbol{\beta}^*(\boldsymbol{z})}{2}$ is convex in $\boldsymbol{z}$. Now, denote the element-wise products $(\boldsymbol{\beta} \cdot \boldsymbol{M})_{ij} = \beta_i M_{ij}$ and $(\boldsymbol{\beta} \cdot \boldsymbol{\mu}) = \beta_i \mu_i$. Then, by direct calculation, the Hessian of $f(\boldsymbol{z})$ in the direction of $\boldsymbol{\beta}$ can be calculated as:

$$
\begin{aligned}
&\boldsymbol{\beta}^\top \frac{\partial f(\boldsymbol{z})}{\partial \boldsymbol{z} \partial \boldsymbol{z}^\top} \boldsymbol{\beta} \\
&= \boldsymbol{\mu}^\top (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot \boldsymbol{M}) (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\mu} \cdot \boldsymbol{\beta}) \\
&- \boldsymbol{\mu}^\top (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot \boldsymbol{M}) (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} ((\boldsymbol{\beta} \cdot \boldsymbol{M}) (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} \boldsymbol{Z}\boldsymbol{\mu}) \\
&= \boldsymbol{\mu}^\top (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot \boldsymbol{M}) (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot (\boldsymbol{I} - \boldsymbol{M}(\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} \boldsymbol{Z}) \boldsymbol{\mu}) \\
&= \frac{1}{\lambda_\beta} \boldsymbol{\mu}^\top (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot \boldsymbol{M}) (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1} \boldsymbol{\mu}) \\
&= \frac{1}{\lambda_\beta} (\boldsymbol{\beta} \cdot (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1} \boldsymbol{\mu})^\top \boldsymbol{M} (\boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} (\boldsymbol{\beta} \cdot (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1} \boldsymbol{\mu}) \\
&= \frac{1}{\lambda_\beta} (\boldsymbol{\beta} \cdot (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1} \boldsymbol{\mu})^\top \boldsymbol{M} (\boldsymbol{M} + \boldsymbol{M}\boldsymbol{Z}\boldsymbol{M})^\dagger \boldsymbol{M} (\boldsymbol{\beta} \cdot (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1} \boldsymbol{\mu}) \\
&\geq 0.
\end{aligned}
$$

where in the second last-step, we utilized:

$$
\boldsymbol{M}(\boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} = \boldsymbol{M}\boldsymbol{M}^\dagger \boldsymbol{M}(\boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1} = \boldsymbol{M}\boldsymbol{M}^\dagger (\boldsymbol{I} + \boldsymbol{M}\boldsymbol{Z})^{-1} \boldsymbol{M} = \boldsymbol{M}(\boldsymbol{M} + \boldsymbol{M}\boldsymbol{Z}\boldsymbol{M})^\dagger \boldsymbol{M}
$$

Since $\boldsymbol{M}, \boldsymbol{Z}$ are both positive semi-definite, it is clear that $(\boldsymbol{M} + \boldsymbol{M}\boldsymbol{Z}\boldsymbol{M})^\dagger$ is positive semi-definite, and thus the Hessian of $f(\boldsymbol{z})$ in the direction of $\boldsymbol{\beta}$ is always non-negative. Since this inequality holds for any $\boldsymbol{\beta}$, the Hessian matrix of $f(\boldsymbol{z})$ is positive semidefinite and hence $f(\boldsymbol{z})$ is convex in $\boldsymbol{z}$. □

Proof of Lemma 3. We begin by differentiating matrix $\boldsymbol{K}$ with respect to $\boldsymbol{Z}$'s diagonal component z_d^t:

$$
\frac{\partial \boldsymbol{K}}{\partial z_d^t} = \frac{\partial \boldsymbol{K}(\boldsymbol{z})}{\partial z_d^t} = \frac{\partial \left(\lambda_\beta \mathbb{I} + \boldsymbol{Z}\boldsymbol{M}\right)}{\partial z_d^t} = \boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}} \boldsymbol{M}. \tag{2.23}
$$

The partial derivative of the inverse of $\boldsymbol{K}$ is then given by

$$\frac{\partial \boldsymbol{K}^{-1}}{\partial z_d^t} = -\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial z_d^t}\boldsymbol{K}^{-1} = -\boldsymbol{K}^{-1}\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{K}^{-1}. \tag{2.24}$$

Finally, we have

$$\frac{\partial c(\boldsymbol{z})}{\partial z_d^t} = \frac{\partial\left(-\frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}\right)}{\partial z_d^t} = \frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{K}^{-1}\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{K}^{-1}\boldsymbol{Z} \;-\; \boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\right)\boldsymbol{\mu}. \tag{2.25}$$

□

Proof of Lemma 4 We first introduce some notation: given any vector (matrix) $\boldsymbol{a}$ ($\boldsymbol{A}$) and a binary vector $\boldsymbol{z}$ (feasible for Problem (2.19)), $\boldsymbol{a}_{\boldsymbol{z}}$ ($\boldsymbol{A}_{\boldsymbol{z},:}$ or $\boldsymbol{A}_{:,\boldsymbol{z}}$) is formed by selecting all entries (t,d) of vector $\boldsymbol{a}$ (all rows (t,d) or all columns (t,d) of matrix $\boldsymbol{A}$, respectively) for which $z_d^t = 1$. Accordingly, the subscript $\boldsymbol{z}^c$ selects the entries/rows/columns for which $z_d^t = 0$.

Cost function evaluation. Define $\boldsymbol{K} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})$. Then given a feasible binary vector $\boldsymbol{z}$, the cost function $c(\boldsymbol{z})$ is $-\frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}$. To evaluate this equation, we first need to invert matrix $\boldsymbol{K}$. The size of matrix $\boldsymbol{K}$ is $TD \times TD$, so a naive implementation would require $O\left(T^3D^3\right)$ operations. We can reduce the complexity of the inversion by exploiting the structure of the matrix as follows:

- We reorder the rows and columns of matrix $\boldsymbol{K}$ so that it takes the form:

$$\tilde{\boldsymbol{K}} := \begin{bmatrix} \lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}} & \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}^c} \\ \boldsymbol{0} & \lambda_\beta \boldsymbol{I} \end{bmatrix},$$

  where $\boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}} \in \mathbb{R}^{TK_{\mathrm{L}} \times TK_{\mathrm{L}}}$ and $\boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}^c} \in \mathbb{R}^{TK_{\mathrm{L}} \times T(D-K_{\mathrm{L}})}$. We then similarly reorder $\boldsymbol{\mu}$ and $\boldsymbol{Z}$ to $\tilde{\boldsymbol{\mu}} = [\boldsymbol{\mu}_{\boldsymbol{z}}, \boldsymbol{\mu}_{\boldsymbol{z}^c}]$ and $\tilde{\boldsymbol{Z}} = \mathrm{Diag}(\boldsymbol{1}_{\boldsymbol{z}}, \boldsymbol{0}_{\boldsymbol{z}^c})$. Note that the reordering does not change the objective value, and therefore the objective function is now $-\frac{1}{2}\tilde{\boldsymbol{\mu}}^\top \tilde{\boldsymbol{K}}^{-1}\tilde{\boldsymbol{Z}}\tilde{\boldsymbol{\mu}}$.

- We perform blockwise inversion, which gives

$$\tilde{\boldsymbol{K}}^{-1} = \begin{bmatrix} (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1} & \frac{1}{\lambda_\beta}(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1}\boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}^c} \\ \mathbf{0} & \frac{1}{\lambda_\beta}\boldsymbol{I} \end{bmatrix}. \tag{2.26}$$

  Since $\boldsymbol{Z}$ has zeros on the diagonals for all $\boldsymbol{z}^c$ columns, we thus have

$$\tilde{\boldsymbol{K}}^{-1}\tilde{\boldsymbol{Z}} = \begin{bmatrix} (\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{2.27}$$

  and therefore the objective function can be now written as $-\frac{1}{2}\tilde{\boldsymbol{\mu}}^\top \tilde{\boldsymbol{K}}^{-1}\tilde{\boldsymbol{Z}}\tilde{\boldsymbol{\mu}} = -\frac{1}{2}\boldsymbol{\mu}_{\boldsymbol{z}}^\top(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1}\boldsymbol{\mu}_{\boldsymbol{z}}$. Noting that the matrix $(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})$ has block tri-diagonal structure, with T blocks of size $K_\mathrm{L} \times K_\mathrm{L}$ each, its inverse $(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1}$ can be computed by recursive application of blockwise inversion in $O\left(T^2 K_\mathrm{L}^2 (T + K_\mathrm{L})\right)$ operations.

- The remaining operations to evaluate the objective are the vector-matrix multiplications with $\boldsymbol{\mu}_{\boldsymbol{z}}$, which require $O(T^2 K_L^2)$ operations.

Gradient evaluation. We compute each of the TD gradient entries as per Lemma 3: $\frac{\partial c(\boldsymbol{z})}{\partial z_d^t} = \frac{1}{2}\boldsymbol{\mu}^\top \boldsymbol{K}^{-1}\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{K}^{-1}\boldsymbol{Z} \;-\; \boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\right)\boldsymbol{\mu}$. $\boldsymbol{v}^i \in \mathbb{R}^{TD}$ denotes auxiliary vectors, and we work as follows:

- We compute $\boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}$. Noting that $\boldsymbol{K}^{-1}\boldsymbol{Z}$ selects the columns (t, d) of $\boldsymbol{K}^{-1}$ for which $z_d^t = 1$ and sets the remaining columns to 0, and observing that $\boldsymbol{K}_{\boldsymbol{z}^c,\boldsymbol{z}}^{-1} = \mathbf{0}$ from Equation 2.26, we in fact only need to compute $\boldsymbol{v}_{\boldsymbol{z}}^{\mathbf{0}} = \boldsymbol{K}_{\boldsymbol{z},\boldsymbol{z}}^{-1}\boldsymbol{\mu}_{\boldsymbol{z}}$. The remaining entries of $\boldsymbol{v}^{\mathbf{0}} \in \mathbb{R}^{TD}$, i.e., $\boldsymbol{v}_{\boldsymbol{z}^c}^{\mathbf{0}}$, are set to 0. This only needs to be performed once independent of which gradient entry is being computed, and we have $\boldsymbol{K}^{-1}$ from the evaluation of the cost function. Thus, the complexity is $O\left(T^2 K_\mathrm{L}^2\right)$ operations.

- We compute $\boldsymbol{\mu}^\top \boldsymbol{K}^{-1}$, namely, $\boldsymbol{v}_{\boldsymbol{z}}^{\mathbf{1}} = \left(\boldsymbol{\mu}_{\boldsymbol{z}}^\top \boldsymbol{K}_{\boldsymbol{z},\boldsymbol{z}}^{-1}\right)^\top$, which requires $O\left(T^2 K_\mathrm{L}^2\right)$ operations. To compute the remaining entries of $\boldsymbol{v}^{\mathbf{1}} \in \mathbb{R}^{TD}$, i.e., $\boldsymbol{v}_{\boldsymbol{z}^c}^{\mathbf{1}}$, we again reorder

$\boldsymbol{\mu}$ to be $\tilde{\boldsymbol{\mu}} := [\boldsymbol{\mu}_{\boldsymbol{z}}, \boldsymbol{\mu}_{\boldsymbol{z}^c}]$ similar to above, and then use the formula indicated in Equation (2.26). Put together, we have

$$\begin{aligned}\boldsymbol{v^1} &= \left[\boldsymbol{\mu}_{\boldsymbol{z}}^\top(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1} \qquad \tfrac{1}{\lambda_\beta}\left(\boldsymbol{\mu}_{\boldsymbol{z}}^\top(\lambda_\beta \boldsymbol{I} + \boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}})^{-1}\boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}^c} + \boldsymbol{\mu}_{\boldsymbol{z}^c}^\top\right)\right] \\ &= \left[\boldsymbol{v}_{\boldsymbol{z}}^{\boldsymbol{1}} \qquad \tfrac{1}{\lambda_\beta}\left(\boldsymbol{v}_{\boldsymbol{z}}^{\boldsymbol{1}}\boldsymbol{M}_{\boldsymbol{z},\boldsymbol{z}^c} + \boldsymbol{\mu}_{\boldsymbol{z}^c}^\top\right)\right], \end{aligned} \tag{2.28}$$

which requires $O\left(T^2 K_{\mathrm{L}} D\right)$ operations. The above steps only need to be performed once, independently of which entry of the gradient is being computed. Overall, the complexity is $O\left(T^2 K_{\mathrm{L}} D\right)$ operations.

- For each (t, d), we compute the multiplication $\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\right)\left(\boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}\right)$. Noting that the multiplication $\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}$ yields a matrix that is nonzero only at row (t, d), and since the result is multiplied with the vector $\boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}$, we implement the multiplication as $\boldsymbol{v}_{(t,d)}^{\boldsymbol{2}} = \boldsymbol{M}_{(t,d),\boldsymbol{z}}\boldsymbol{v}_{\boldsymbol{z}}^{\boldsymbol{0}}$. This requires $O\left(T^2 D K_{\mathrm{L}}\right)$ operations in total across all (t, d).

- For each (t, d), we compute the multiplication $\left(\boldsymbol{\mu}^\top\boldsymbol{K}^{-1}\right)\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}\right)$. We first note that $\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}} = \boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}$ and hence the multiplication can be rewritten as: $\left(\boldsymbol{\mu}^\top\boldsymbol{K}^{-1}\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\right)\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}\right) = \left(\boldsymbol{\mu}^\top\boldsymbol{K}^{-1}\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\right)\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{v}^2\right)$. Therefore, the first term selects the (t, d) column of $\boldsymbol{\mu}^\top\boldsymbol{K}^{-1}$ and the second term selects the (t, d) row of $\boldsymbol{v}^2$. Using this fact, along with Equation (2.28), we can now go back and calculate the final product $\boldsymbol{v}_{(t,d)}^{\boldsymbol{3}} = \left(\boldsymbol{\mu}^\top\boldsymbol{K}^{-1}\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\right)\left(\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{M}\boldsymbol{K}^{-1}\boldsymbol{Z}\boldsymbol{\mu}\right) = \boldsymbol{v}_{(t,d)}^{\boldsymbol{1}} \cdot \boldsymbol{v}_{(t,d)}^{\boldsymbol{2}}$. The complexity is $O\left(TD\right)$ operations in total across all (t, d).

- For each (t, d), we compute the multiplication $\boldsymbol{\mu}^\top\boldsymbol{K}^{-1}\boldsymbol{E}_{\boldsymbol{d}}^{\boldsymbol{t}}\boldsymbol{\mu}$ as $\boldsymbol{v}_{(t,d)}^{\boldsymbol{4}} = \boldsymbol{v}_{(t,d)}^{\boldsymbol{1}} \cdot \boldsymbol{\mu}_{(t,d)} + \mathbb{1}_{\left(z_d^t = 0\right)}\frac{\left(\boldsymbol{\mu}_{(t,d)}\right)^2}{\lambda_\beta}$. This requires $O\left(TD\right)$ operations in total across all (t, d).

- For each (t, d), we compute ($O\left(TD\right)$ operations in total across all (t, d)) the corresponding entry of the gradient as $\frac{\partial c(\boldsymbol{z})}{\partial z_d^t} = \frac{\boldsymbol{v}_{(t,d)}^{\boldsymbol{3}} - \boldsymbol{v}_{(t,d)}^{\boldsymbol{4}}}{2}$.

After completing the steps outlined above, we have the ingredients to compute all entries of the gradient $\nabla_{\boldsymbol{z}} c(\boldsymbol{z})$. In total, the cost is $O\left(T^2 D K_{\mathrm{L}}\right)$ operations.

Cut generation. The complexity of the entire process is $O\left(T^2 K_{\mathrm{L}}[K_{\mathrm{L}}(T+K_{\mathrm{L}})+D]\right)$. □

Proof of Lemma 5. Let us denote by $\mathcal{Z}_\beta$ the feasible set defined by Equations (2.2)-(2.4). We upper bound Problem (2.1)-(2.4) as follows:

$$\min_{\boldsymbol{\beta}\in\mathcal{Z}_\beta} \quad \sum_{n=1}^{N}\sum_{t=1}^{T}\left(y_n^t - \sum_{d=1}^{D} X_{n,d}^t \beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\sum_{d=1}^{D}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{(s,t)\in E}\sum_{d=1}^{D}\left(\beta_d^t - \beta_d^s\right)^2 \tag{2.29}$$

$$\leq \frac{1}{D}\sum_{d=1}^{D} \quad \min_{\boldsymbol{\beta}_d} \quad \sum_{n=1}^{N}\sum_{t=1}^{T}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{(s,t)\in E}\left(\beta_d^t - \beta_d^s\right)^2 \tag{2.30}$$

$$= \min_{\boldsymbol{\beta}} \quad \frac{1}{D}\sum_{d=1}^{D}\left(\sum_{n=1}^{N}\sum_{t=1}^{T}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{(s,t)\in E}\left(\beta_d^t - \beta_d^s\right)^2\right) \tag{2.31}$$

$$\leq \min_{\boldsymbol{\beta}\in\mathcal{Z}_\beta} \quad \frac{1}{D}\sum_{n=1}^{N}\sum_{t=1}^{T}\sum_{d=1}^{D}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\sum_{d=1}^{D}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{(s,t)\in E}\sum_{d=1}^{D}\left(\beta_d^t - \beta_d^s\right)^2 \tag{2.32}$$

$$\leq \min_{\boldsymbol{\beta}\in\mathcal{Z}_\beta} \quad \frac{1}{D}\sum_{n=1}^{N}\sum_{t=1}^{T}\sum_{d=1}^{D}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\sum_{d=1}^{D}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{(s,t)\in E}\sum_{d=1}^{D} 2[\left(\beta_d^t\right)^2 + \left(\beta_d^s\right)^2] \tag{2.33}$$

$$= \min_{\boldsymbol{\beta}\in\mathcal{Z}_\beta} \quad \frac{1}{D}\sum_{n=1}^{N}\sum_{t=1}^{T}\sum_{d=1}^{D}\left(y_n^t - X_{n,d}^t\beta_d^t\right)^2 + \lambda_\beta \sum_{t=1}^{T}\sum_{d=1}^{D}\left(\beta_d^t\right)^2 + \lambda_\delta \sum_{t=1}^{T}\sum_{d=1}^{D} 2d^t\left(\beta_d^t\right)^2. \tag{2.34}$$

For the **first inequality**, in (2.29) we have the prediction error of the best multivariate model, which by definition is less than or equal to the error of any univariate model. This can therefore be upper bounded by the average error among all univariate models, which is what we have in (2.30). Observe that the best among these univariate models is indeed feasible for the minimization problem in (2.29). The **equality** between (2.30) and 2.31 is due to separability. For the **second inequality**, in 2.31 we have

an unconstrained problem, whereas, in 2.32 we require that $\boldsymbol{\beta} \in \mathcal{Z}_\beta$ hence restricting the feasible set. Moreover, in 2.32 we rescale the regularization term and the slowly varying penalty with D so that their relative importance compared to the prediction error is in the same order as in the original problem. For the **third inequality**, we trivially bound the squares of the differences between coefficients in adjacent vertices. □

Proof of Proposition 2. The termination condition of Algorithm 2 guarantees that, at termination, the solution must be a feasible solution for Problem (2.1)-(2.4). Therefore, we only need to prove that Algorithm 2 terminates in polynomial time.

The first step in Algorithm 2 involves computing the loss for each vertex-feature pair. This requires solving TD univariate regularized least squares problems, and can be done in closed form in time $O(NTD)$.

The second step in Algorithm 2 involves solving a linear optimization problem over TD variables. This can be done in time $\tilde{O}((TD)^{2+1/6})$ using the algorithm by [81]. (We note that $\tilde{O}(\cdot)$ gives the asymptotic complexity ignoring logarithmic factors.)

The third step in Algorithm 2 involves ensuring integrality by iterating over all entries of the linear optimization problem's solution, and can be done in time $O(TD)$.

The fourth step in Algorithm 2 involves ensuring feasibility by removing one feature at a time as long as the linear optimization problem's solution is infeasible. Let us denote by S the global support (across all vertices) of the estimated regression coefficients. Since, after each iteration, we remove one feature from S, the global sparsity constraint (2.3) is guaranteed to be satisfied after at most $D - K_\mathrm{G}$ iterations. Similarly, after at most $D - (K_\mathrm{L} + \frac{K_\mathrm{C}}{2})$ iterations, all vertices will be constrained to include the same set of $K_\mathrm{L} + \frac{K_\mathrm{C}}{2}$ features and hence any pair of similar regressions will differ in at most K_C features. Therefore, the while loop terminates in at most

$$\max\{D - K_\mathrm{G}, D - (K_\mathrm{L} + \frac{K_\mathrm{C}}{2})\}$$

iterations, which gives an asymptotic complexity of $O(D)$. The complexity of each iteration is $O(T)$: in an efficient implementation, the first two steps inside the while

loop are performed once, and, in each iteration, we only update the corresponding data structures in $O(1)$ time. Therefore, the fourth step can be done in time $O(TD)$.

The fifth step in Algorithm 2 involves computing $\boldsymbol{\beta}^H$ for the estimated support $\boldsymbol{z}^H$. This can be done in time $O(T^2K_{\mathrm{L}}^2(T+K_{\mathrm{L}}))$ using the procedure described in the proof of Lemma 4.

Therefore, Algorithm 2 terminates in polynomial time and returns a feasible solution of the original problem. □

2.10 Extended Experiments

2.10.1 Algorithms and Software

In this section, we give the implementation details of the algorithms which we compare in our experiments. For a fair comparison, we implement all algorithms in `Julia` programming language (version 1.6) and using the `JuMP.jl` modeling language for mathematical optimization (version 0.21). We solve the optimization models using the `Gurobi` commercial solver (version 9.5). All experiments were performed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running CentOS release 7. We make our code available at `https://github.com/vvdigalakis/SSVRegression.git`.

We consider the following algorithms:

- *Sparse regression:* We fit a single (static) sparse regression model across all vertices. Note that, as a result, this approach uses $N' = NT$ data points to train D parameters (since the same set of parameters is estimated across all vertices). We solve the sparse regression formulation, as shown in Problem (2.6), using the cutting plane algorithm by [35] and the `Gurobi` solver. We refer to this approach as `sparse_regression`.

- *Sum-of-norms regularization:* We fit a slowly varying regression model in which penalize the sum across all pairs of adjacent vertices of the ℓ_p difference, for $p \in \{0, 1\}$, between the corresponding coefficients, as shown in Problem (2.5) [178]. In the $p = 1$ case, the resulting problem can be reformulated as a

quadratic optimization problem. In the $p = 2$ case, the resulting problem can be reformulated as a second-order cone optimization problem. In both cases, we directly solve the resulting problems using `Gurobi`. We refer to this approach as `sum_of_norms_lp`.

- *Sum-of-norms and lasso regularization:* We expand the `sum_of_norms_lp`approach with an ℓ_1 penalty on the coefficients to add robustness and -hopefully- encourage some level of sparsity. We again reformulate the resulting problem as linear or second-order cone optimization problem, which we solve using `Gurobi`. We refer to this approach as `sum_of_norms_lp_lasso`.

- *SSVR via the heuristic algorithm:* We implement Algorithm 2 using the `Gurobi` solver. We refer to this approach as `svar_heuristic`.

- *SSVR via the exact cutting plane algorithm:* We implement Algorithm 1 using the `Gurobi` solver. We refer to this approach as `svar_cutplane`.

For all methods, we impose a time limit of 900 seconds; if no solution is returned when the solver terminates, we return the all-zeros solution. We remark that the solver may not stop immediately upon hitting the time limit; it will instead stop after performing the required additional computations of the attributes associated with the terminated optimization [124]. Moreover, if the solution time of a method exceeded 1 hour in preliminary experiments, we did not include this method in our reported experiments.

Each of the above models is hyperparameter tuned using holdout validation and exhaustive grid search over the cross product of the selected ranges of values of all hyperparameters. Specifically, we consider 3 values for λ_β, starting at N and decreasing by a factor of 2 to obtain each next value, and 3 values for λ_δ, starting at $\sqrt{N}$ and decreasing by a factor of 2 to obtain each next value. For `svar_cutplane`, `svar_heuristic`, and `sparse_regression`, we estimate the final coefficients using a regularization weight of $\sqrt{\lambda^\star_\beta}$, where $\lambda^\star_\beta$ is the regularization weight selected through the validation process; we empirically observe that such an approach slightly improves the performance of these methods. In all synthetic experiments, except for the ones

presented in Section 2.7.4, we assume that the sparsity-related hyperparameters are known.

2.10.2 Experiments on Synthetic Data: Detailed Methodology

In this section, we provide the details of the data generation and evaluation methodology we use in our synthetic data experiments in Section 2.7.

Ground truth coefficients. We generate a matrix of ground truth coefficients $\boldsymbol{\beta} \in \mathbb{R}^{T \times D}$. Each element $\boldsymbol{\beta}^t \in \mathbb{R}^D$ is the vector of coefficients of the regression at vertex $t \in [T]$. We focus on the spatially varying case, where the similarity graph is a general graph, as the temporally varying case is essentially a special case. To generate $\boldsymbol{\beta}$, we control the parameters presented in Table 2.4.

Table 2.4: Data generation parameters.

Parameter	Explanation
$K_{\mathrm{L}} \in \mathbb{Z}^+$	Local sparsity, as detailed in Equation (2.2).
$K_{\mathrm{G}} \in \mathbb{Z}^+$	Global sparsity, as detailed in Equation (2.3).
$K_{\mathrm{C}} \in \mathbb{Z}^+$	Number of changes in support, as detailed in Equation (2.4).
$\sigma_v \in [0, 1]$	Maximum % of change in coefficients between similar vertices; drawn uniformly at random from $[-\sigma_v, +\sigma_v]$.
$d_G \in \mathbb{R}^+$	Similarity graph density.
$\rho_d \in [0, 1]$	Correlation across features.
$\xi \in \mathbb{R}^+$	Signal-to-noise ratio for the noise added to the outcome variable.

Given the data generation parameters, the actual generation of $\boldsymbol{\beta}$ is as follows. We generate a random Erdos-Renyi (ER) graph G with $d_G \frac{(T-1)\log T}{2}$ edges. The rationale behind this value is the following:

- Consider a random graph G drawn according the ER model where each edge is included in G with probability $p = d_G \frac{\log T}{T}$, independently from every other edge.
- The expected number of edges is then $p\binom{T}{2} = \frac{d_G T(T-1)\log T}{2T}$.
- Noting that $p = \frac{\log T}{T}$ is a sharp threshold for connectedness of G, by setting $d_G > 1$, the resulting graph will almost surely be connected, whereas, by setting $d_G < 1$, the resulting graph will almost surely be disconnected.

- In our experiments, we would like to directly control the number of edges in G, so we instead sample G uniformly at random from the collection of all graphs which have $d_G \frac{(T-1)\log T}{2}$ edges.

Therefore, d_G controls the density and connectedness of G.

We then randomly choose the global support S according to the desired value of K_{G}, i.e., $|S| = K_{\mathrm{G}}$. For each connected component C of G, we generate an initial vector of coefficients $\boldsymbol{\beta}^C$, satisfying the local sparsity constraint, uniformly at random from $\{-1, 1\}^{K_{\mathrm{L}}}$ (note that we allow only features from the global support to be selected). Then, for each vertex $t \in C$, we construct $\boldsymbol{\beta}^t$ by perturbing $\boldsymbol{\beta}^C$ according to the desired σ_v. The desired number of changes in support is performed by randomly replacing features which originally were in the support, with features which were not, at randomly selected vertices from the global support S.

Design Matrix and Response. We create the design matrix $\boldsymbol{X} \in \mathbb{R}^{N \times T \times D}$ as follows. We assume that, for $t \in [T]$, $\boldsymbol{X}^t = (\boldsymbol{x}_1^t, \ldots, \boldsymbol{x}_N^t)$ are i.i.d. realizations from a D-dimensional zero-mean normal distribution with covariance matrix $\boldsymbol{\Sigma}$, i.e., $\boldsymbol{x}_n^t \sim \mathcal{N}(\mathbf{0}_D, \boldsymbol{\Sigma}), n \in [N]$. The covariance matrix $\boldsymbol{\Sigma}$ is parameterized by the correlation coefficient $\rho_d \in [0, 1]$ as $\Sigma_{ij} = \rho_d^{|i-j|}, \forall i, j \in [D]$. As $\rho_d \to 1$, the columns of the data matrix $\boldsymbol{X}^t$, i.e., the features, become more alike.

The outcome vectors $\boldsymbol{Y} \in \mathbb{R}^{N \times T}$ are created by applying $\boldsymbol{\beta}$ on $\boldsymbol{X}$ and adding i.i.d. noise drawn from a normal distribution $\mathcal{N}(0, \sigma^2)$ to each entry in $\boldsymbol{Y}$, where σ^2 is selected to satisfy $\xi^2 = \frac{\sum_{t \in T} \|\boldsymbol{X^t \beta^t}\|^2}{\sigma^2}$ according to the desired signal-to-noise ratio ξ.

Evaluation Tasks and Metrics. Our task is to estimate $\boldsymbol{\beta}$ and make out-of-sample predictions for unseen data $\boldsymbol{X}_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times T \times D}$ and $\boldsymbol{Y}_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times T}$, generated according to the same process as $\boldsymbol{X}$ and $\boldsymbol{Y}$. We consider the evaluation metrics shown in Table 2.5. We perform a full sensitivity analysis with respect to the problem parameters $(N, T, D, K_{\mathrm{L}}, K_{\mathrm{G}}, K_{\mathrm{C}}, \sigma_v, d_G, \rho_d, \xi)$. For each problem parameter setting, we independently generate 10 datasets and report the mean and standard deviation of the results for each evaluation metric.

Table 2.5: Evaluation Metrics.

Metric	Explanation
MAE	Mean absolute error in estimated coefficients.
ACC	Support recovery accuracy (percentage of truly relevant feature found).
FA	Support recovery false alarm rate (percentage of features found that are irrelevant).
Test R^2	Out-of-sample R^2 statistic (evaluated on held-out test set).
Time	Computational time (in seconds). Measures time to refit after any hyperparameter tuning.
Gap	Optimality gap for MIO-based methods.
Cut Count	Number of cuts generated by cutting plane method.
ACT	Average time per cut generated by cutting plane method.

2.10.3 Experiments on Real-World Data: Methodology and Extended Results

In this section, we provide a more detailed discussion on our computational study on real-world data. First, we give more information on the datasets and the preprocessing methodology we apply to each of them. Then we present the detailed computational results for each dataset, method, and metric combination, which we use to extract the aggregated results shown in Table 2.3.

Datasets and Preprocessing Methodology

We begin our discussion by outlining the details of the real-world datasets we use in our experiments and the preprocessing methodology we apply to each of them.

As discussed in Section 2.8, we randomly split each dataset 10 times into training (60%), validation (20%), and test (20%) sets (respecting the temporal structure if such exists). In all cases, we use the training set to normalize both the validation and the test sets' data matrices $\boldsymbol{X}$ and responses $\boldsymbol{Y}$, so that all features and responses have zero mean and unit variance.

Given that all hyperparameters are now unknown, we tune each method's hyperparameters using holdout validation and exhaustive grid search over a total of $5 - 15$ hyperparameter combinations. Specifically, for `svar_cutplane` and `svar_heuristic`, we construct the validation grid as the cross-product of 3 values for K_{L}, 2 values for K_{G}, and 2 values for K_{C}, and set $\lambda_\beta = N$ and $\lambda_\delta = \sqrt{N}$, which were shown to perform well in our synthetic data experiments; for `sparse_regression`, we construct

the validation grid as 5 values for K_{L}, and set $\lambda_\beta = N$; for `sum_of_norms_l1` and `sum_of_norms_l2`, we construct the validation grid as 3 values for λ_δ, starting at $\sqrt{N}$ and decreasing by a factor of 2 to obtain each next value; for `sum_of_norms_l1_lasso` and `sum_of_norms_l2_lasso`, we construct the validation grid as the cross-product of the grid we use for `sum_of_norms_l1` and `sum_of_norms_l2`, and 5 values λ_β, starting at N and decreasing by a factor of 2 to obtain each next value.

Appliances Energy Prediction: Hourly. In this experiment, we focus on a real-world case study concerned with appliances energy prediction [70]. The dataset is publicly available at the University of California Irvine (UCI) Machine Learning repository, at `https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction`.

Each observation in the dataset is a vector of measurements made by a wireless sensor network in a low energy building. The features include the temperature and humidity conditions in various rooms in the building, the weather conditions in the nearest weather station, the month at which the measurements were taken, and a couple of noise variables. The goal is to predict the energy consumption of the building's appliances. Measurements are taken every 10 minutes over a 4.5-month period.

We preprocess the dataset as follows. We construct the similarity graph by assigning a vertex to each hour of the day, so that $T = 24$. To capture the temporal structure in the problem, the graph is a chain, i.e., vertex $t \in [T-1]$ is considered adjacent to vertex $t+1$. For each day d in the data, we create 6 data points per vertex $t \in [T]$, by collecting all 6 measurements that were taken at hour t and during day d. For example, for $t = 15$, we collect the measurements taken at 3pm, 3:10pm, ..., 3:50pm, across all days in the data. By doing so, we get $N = 822$ data points per vertex. Each data point consists of $D = 26$ features. The decision of splitting the data hourly was the most natural, but it remains an arbitrary decision. The model can be applied to any subdivision depending on the goal of the regression, and this splitting can also be hyper-parameter tuned for further performance improvement.

Appliances Energy Prediction: Monthly. In this experiment, we consider the same appliances energy prediction dataset. However, instead of assigning a vertex to each hour of the day, we now assign a vertex to each month in the data, so that $T = 5$ (the data covers a 4.5-month period between January and May). For each measurement, we replace in the feature set the month with the hour at which the measurement was taken. Once again, to capture the temporal structure in the problem, we take the similarity graph to be a chain. We now collect all measurements taken during each month as data points for the corresponding vertex, and subsample $N = 2,922$ data points so that we get the same N across all months.

Housing Price Prediction. In this experiment, we explore the application of our framework to the task housing price prediction, in Ames, Iowa [87]. The dataset is publicly available at `http://jse.amstat.org/v19n3/decock/DataDocumentation.txt`.

The original dataset contains $2,930$ observations and a large number of features (23 nominal, 23 ordinal, 14 discrete, and 20 continuous) involved in assessing home values. The sales took place in Ames, Iowa, from 2006 to 2010. The goal is to predict the price at which the house was sold.

We preprocess the dataset as follows. We first drop features with missing values in over 1% of the observations; then we drop any observation that still has missing features. We use one-hot encoding for the nominal features and integer encoding for ordinal and discrete variables. Each data point consists of $D = 199$ features. The dataset contains information on the neighborhood where each house is located, so we could have used these neighborhoods as the vertices in the similarity graph. Nevertheless, such an approach leads to highly imbalanced vertices in terms of the number of data points that fall therein (due to the fact that many sales were performed at some neighborhoods and very few at others). To address this issue, we cluster the neighborhoods into larger groups, while requiring that neighborhoods that fall into the same group be adjacent and that the number of data points that fall into each group be relatively balanced. Then we construct the similarity graph by adding an

edge between groups of neighborhoods that are adjacent. In the end, we obtain $T = 7$ groups of neighborhoods, each with at least $N = 352$ data points (for simplicity, we randomly select exactly $N = 352$ data points in each group). The similarity graph consists of $E = 8$ edges.

Air Quality. In this experiment, we consider the task of air quality prediction, in Beijing [237]. The dataset is publicly available at `https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data`.

The original dataset consists of 420,768 observations and features: 5 numerical features (temperature, pressure, dew point temperature, precipitation, wind speed), 1 categorical feature (wind direction), and 3 time-related features. The goal is to predict PM2.5 concentration - an air pollutant that is a concern for people's health when levels in air are high. The data is collected from 12 nationally-controlled air quality monitoring sites. The meteorological data in each air quality site are matched with the nearest weather station from the China Meteorological Administration. The time period is from March 1st, 2013, to February 28th, 2017.

We preprocess the dataset as follows. We construct the similarity graph by assigning a vertex to each air quality monitoring station, so that $T = 12$. We add an edge between each station and the closest station towards each direction (east, north, south, west), provided that their distance does not exceed a pre-defined threshold, for a total of $E = 14$ edges. The resulting similarity graph is disconnected and consists of 4 connected components. We get $N = 35,064$ data points per vertex. After one-hot encoding the wind direction categorical feature into 17 binary features, we get a total of $D = 25$ features. We finally perform mean imputation.

Meteorology. In this experiment, we consider the task of weather prediction. The dataset is publicly available at `https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data`.

The original dataset contains about 5 years of hourly measurements of various weather attributes, including temperature, humidity, air pressure, wind direction, and

wind speed. The goal is to predict the temperature half a day in advance. This data is collected from 30 US and Canadian Cities, as well as 6 Israeli cities.

We preprocess the dataset as follows. We construct the similarity graph by assigning a vertex to each city, so that $T = 36$. We add an edge between two cities provided that their (euclidean) distance is less than $1,000$ kilometers, for a total of $E = 110$ edges. The resulting similarity graph is disconnected and consists of 2 connected components. We get $N = 45,231$ data points per vertex. Each data point consists of measurements of the 5 aforementioned weather attributes over the past 10 hours, for a total of $D = 50$ features. We note that we made a number of arbitrary decisions in preprocessing the data, including predicting half a day ahead (predicting fewer hours ahead led to extremely high R^2 simply by outputting the current temperature value), setting the distance threshold to 1,000 kilometers, and considering the past 10 measurements for each weather attribute; in all cases, we set the above values to what seemed the most natural choice.

Experiments on Real-World Data: Extended Results

In this section, we provide extended computational results from our experiments on real-world data. In particular, for each dataset-method pair, we give the mean and standard deviation of each metric discussed in the aggregated results of Table 2.3.

Table 2.6 summarizes, for each dataset-method pair, the averaged (across all 10 training-validation-test splits) out-of-sample R^2 results. Here, `sum_of_norms_l1` and `sum_of_norms_l2` perform best (although `sum_of_norms_l2` still faces serious scalability issues), closely followed by `svar_cutplane`. We note that `svar_cutplane` usually improves upon the solution found by `svar_heuristic`, and that `svar_heuristic` performs surprisingly well, outperforming `sparse_regression` and the lasso-based regularized methods. The benefits that sparsity can provide on generalization can be seen through the housing case study, where `sum_of_norms_l1` and `sum_of_norms_l1_lasso` produce much denser models, which fail to generalize out-of-sample.

In Table 2.7, we assess, for each dataset-method pair, the learned models' interpretability, through the (average) estimated sparsity-related hyperparameters $\hat{K}_{\mathrm{L}}$, $\hat{K}_{\mathrm{G}}$,

Table 2.6: Real-world data experiments: out-of-sample R^2.

Algorithm	Air Quality	Appliances Energy (hour)	Appliances Energy (month)	Housing Price	Meteorology
svar_cutplane	0.5179 (0.0)	0.8444 (0.03)	0.5043 (0.02)	0.9591 (0.01)	0.8428 (0.0)
svar_heuristic	0.5174 (0.0)	0.844 (0.03)	0.4944 (0.02)	0.9587 (0.0)	0.8411 (0.0)
sparse_regression	0.5151 (0.0)	0.6103 (0.02)	0.4544 (0.02)	0.9402 (0.01)	0.8302 (0.0)
sum_of_norms_l1	0.5192 (0.0)	0.8454 (0.03)	0.5047 (0.02)	0.6538 1.0)	0.8449 (0.0)
sum_of_norms_l1_lasso	0.4986 (0.0)	0.8415 (0.03)	0.4877 (0.03)	0.647 0.99)	0.6474 0.58)
sum_of_norms_l2	0.5191 (0.0)	0.845 (0.03)	0.506 (0.02)	-	-
sum_of_norms_l2_lasso	0.4987 (0.0)	0.842 (0.03)	0.4896 (0.02)	-	-

$\hat{K}_{\mathrm{C}}$. The edge of `svar_cutplane` and `svar_heuristic` among the slowly varying methods is evident: the learned models are significantly sparser and hence more interpretable, while achieving comparable or even improved predictive performance.

Table 2.7: Real-world data experiments: model sparsity ($\hat{K}_{\mathrm{L}}$, $\hat{K}_{\mathrm{G}}$, $\hat{K}_{\mathrm{C}}$).

Algorithm	Air Quality	Appliances Energy (hour)	Appliances Energy (month)	Housing Price	Meteorology
svar_cutplane	7.3 (0.48) | 8.2 (1.93) | 3.6 (6.72)	19.4 (1.26) | 23.1 (2.18) | 24.6 (6.93)	22.0 (2.0) | 23.5 (2.59) | 4.4 (3.2)	46.5 (4.12) | 61.4 (8.06) | 77.0 (24.89)	40.0 (0.0) | 45.4 (0.52) | 70.5 (26.17)
svar_heuristic	7.2 (0.42) | 7.9 (1.66) | 2.0 (3.53)	19.1 (1.45) | 21.0 (2.11) | 16.8 (2.86)	23.0 (0.0) | 25.3 (0.82) | 5.4 (1.35)	46.5 (4.12) | 58.6 (5.25) | 67.2 (21.84)	40.0 (0.0) | 45.0 (0.0) | 50.0 (0.0)
sparse_regression	6.1 (0.32) | 6.1 (0.32) | 0.0 (0.0)	22.5 (3.31) | 22.5 (3.31) | 0.0 (0.0)	24.6 (0.84) | 24.6 (0.84) | 0.0 (0.0)	50.0 (4.08) | 50.0 (4.08) | 0.0 (0.0)	45.0 (0.0) | 45.0 (0.0) | 0.0 (0.0)
sum_of_norms_l1	25.0 (0.0) | 25.0 (0.0) | 0.0 (0.0)	28.0 (0.0) | 28.0 (0.0) | 0.0 (0.0)	28.0 (0.0) | 28.0 (0.0) | 0.0 (0.0)	173.1 (60.86) | 173.1 (60.86) | 0.0 (0.0)	50.0 (0.0) | 50.0 (0.0) | 0.0 (0.0)
sum_of_norms_l1_lasso	16.3 (0.67) | 20.0 (0.94) | 30.0 (6.41)	14.6 (2.07) | 26.4 (1.17) | 117.2 (17.59)	15.0 (1.15) | 19.7 (1.06) | 27.8 (2.86)	66.1 (29.37) | 116.0 (46.77) | 258.1 (118.63)	23.3 (8.54) | 35.9 (12.97) | 589.4 (207.45)
sum_of_norms_l2	25.0 (0.0) | 25.0 (0.0) | 0.0 (0.0)	28.0 (0.0) | 28.0 (0.0) | 0.0 (0.0)	28.0 (0.0) | 28.0 (0.0) | 0.0 (0.0)	-	-
sum_of_norms_l2_lasso	22.2 (3.65) | 24.2 (1.14) | 45.4 (22.35)	27.8 (0.63) | 28.0 (0.0) | 66.2 (48.23)	22.0 (3.89) | 26.4 (1.58) | 34.7 (15.97)	-	-

Table 2.8 reports, for each dataset-method pair, the corresponding average computational time in seconds for each dataset-method pair. `svar_heuristic` is again the clear winner solving 4 out of 5 problems in milliseconds. The MIO-based methods time out without proving optimality only in one case.

Table 2.8: Real-world data experiments: computational time (in seconds).

Algorithm	Air Quality	Appliances Energy (hour)	Appliances Energy (month)	Housing Price	Meteorology
svar_cutplane	20.89 (21.12)	13.63 (11.05)	0.3 (0.22)	900.73 (0.07)	6.3 (0.63)
svar_heuristic	0.61 (0.08)	0.64 (0.05)	0.21 (0.01)	0.34 (0.05)	6.63 (0.8)
sparse_regression	5.85 (2.39)	0.97 (1.35)	0.33 (0.1)	900.3 (0.07)	343.25 (189.51)
sum_of_norms_l1	87.42 (6.18)	5.52 (0.09)	3.06 (0.03)	19.32 (5.44)	1521.85 (131.2)
sum_of_norms_l1_lasso	86.43 (6.31)	5.5 (0.08)	3.07 (0.04)	107.96 (284.08)	1794.9 (675.08)
sum_of_norms_l2	90.42 (5.96)	17.97 (1.8)	3.73 (0.1)	-	-
sum_of_norms_l2_lasso	90.66 (6.32)	20.36 (1.49)	3.82 (0.07)	-	-

Finally, in Table 2.9, we compare, on real-world data, the proposed cutting plane algorithm (Algorithm 1) with the cutting plane algorithm of [35]. In all but one (housing) case studies, both methods are able to prove optimality; even in the housing case study, `svar_cutplane` achieves a lower optimality gap. Moreover, in 4 out of 5 cases, `svar_cutplane` generates fewer cuts, while both methods' average cut generation times are very small (usually below 0.05 seconds).

Table 2.9: Real-world data experiments: evaluation of the cutting plane method (Gap, Cut Count, ACT).

Algorithm	Air Quality	Appliances Energy (hour)	Appliances Energy (month)	Housing Price	Meteorology
svar_cutplane	0.0 (0.0) | 179.1 (89.32) | 0.0 (0.0)	0.0 (0.0) | 64.3 (30.36) | 0.03 (0.0)	0.0 (0.0) | 13.0 (6.88) | 0.0 (0.0)	0.73 (0.31) | 851.0 (109.05) | 0.06 (0.01)	0.0 (0.0) | 9.0 (1.05) | 0.27 (0.03)
sparse_regression	0.0 (0.0) | 164.2 (50.45) | 0.03 (0.01)	0.0 (0.0) | 145.6 (213.92) | 0.0 (0.0)	0.0 (0.0) | 41.9 (28.66) | 0.0 (0.0)	1.0 (0.0) | 6422.4 (662.54) | 0.0 (0.0)	0.0 (0.0) | 460.2 (158.37) | 0.73 (0.19)

2.10.4 Experiments for Section 2.4: Testing Different Relaxations

In this section, we test the performance of the following (extended) exact convex relaxation of the closed form solution given in Equation (2.17):

$$(\boldsymbol{Z}(\boldsymbol{M}+\lambda_\beta \boldsymbol{I})\boldsymbol{Z})^\dagger \boldsymbol{Z} = (\lambda_\beta \boldsymbol{I} + \boldsymbol{Z}\boldsymbol{M})^{-1}\boldsymbol{Z} + \mu\left(\sum_{t=1}^{T}\sum_{d=1}^{D}\left(z_d^t - \frac{1}{2}\right)^2 - \frac{TD}{4}\right), \quad (2.35)$$

for $\mu \in \{0, 0.1, 1, 2, 5\}$, using synthetic data.

Specifically, we set $N = 1000$, $T = 1$, $D = 100$, $K_{\mathrm{L}} = 5$, $\lambda_\beta = 10$, hence focusing on the standard sparse regression problem. We generate data of the form $\boldsymbol{y} = \boldsymbol{X}_0\boldsymbol{\beta}_0 + \boldsymbol{\varepsilon}$, where $\boldsymbol{X}_0 \in \mathbb{R}^{N\times K_L}$ and $(\boldsymbol{X}_0)_{n,d} \sim N(0,1)$, $\boldsymbol{\beta}_0 \in \mathbb{R}^{K_L}$ and $(\boldsymbol{\beta}_0)_d \sim N(0,1)$, $\boldsymbol{\varepsilon} \in \mathbb{R}^N$ and $(\boldsymbol{\varepsilon})_n \sim N(0, 0.1)$, and we set $\boldsymbol{X} = [\boldsymbol{X}_0, \boldsymbol{Z}]$, where $\boldsymbol{Z} \in \mathbb{R}^{N\times(D-K_L)}$ and $(\boldsymbol{Z})_{n,d} \sim N(0,1)$. We generate 20 instances of the synthetic data.

For each instance, we run 5 versions of Algorithm 1: in each version, we solve the inner problem using Equation (2.35) and a different value for $\mu \in \{0, 0.1, 1, 2, 5\}$. We record the computational time in seconds and the mean absolute error (MAE) in the estimated coefficients, as detailed in Table 2.10. We observe that as μ increases, both the computational time and MAE suffer, suggesting that such family of relaxations is unlikely to produce stronger cuts than the baseline $\mu = 0$ relaxation.

Table 2.10: Results for extended convex relaxations.

μ	Time	MAE
0	0.19	0.056
0.1	0.27	0.069
1	0.97	0.128
2	1.61	0.112
5	10.60	0.150

2.10.5 Experiments for Section 2.6: Integrality Test of Algorithm 2

In this section, we test the integrality of the solutions obtained by the second step of Algorithm 2, that is, the linear relaxation of Problem (2.22), in the temporal case with T time periods (see Figure 2-1), and using synthetic data.

Note that, in this setting, the linear relaxation that Algorithm 2 solves can be written as:

$$
\begin{aligned}
\min_{\boldsymbol{z},\boldsymbol{s},\boldsymbol{w}} \quad & \sum_{t\in[T],d\in[D]} L_d^t z_d^t & \\
\text{s.t.} \quad & z_d^t \le s_d & \forall t \in [T],\ d \in [D] \\
& z_d^{t+1} - z_d^t \le w_d^t & \forall t \in [T-1],\ d \in [D] \\
& z_d^t - z_d^{t+1} \le w_d^t & \forall t \in [T-1],\ d \in [D] \\
& \sum_{d\in[D]} s_d \le K_G & \forall t \in [T],\ d \in [D] \\
& \sum_{d\in[D]} z_d^t = K_L & \forall t \in [T] \\
& \sum_{t\in[T-1],d\in[D]} w_d^t \le K_C & \forall t \in [T-1],\ d \in [D] \\
& 0 \le z_d^t \le 1 & \forall t \in [T],\ d \in [D] \\
& 0 \le w_d^t \le 1 & \forall t \in [T-1],\ d \in [D] \\
& 0 \le s_d \le 1 & \forall t \in [T],\ d \in [D]
\end{aligned}
$$

We test 3 values for each parameter, namely, $T, D \in \{2, 5, 10\}$, for a total of 9 combinations. For each combination, we generate the remaining parameters as follows:

- K_L is selected uniformly within $[\lfloor D/4 \rfloor, \lfloor D/2 \rfloor]$.
- K_G is selected uniformly within $[\lfloor 1.5K_L \rfloor, \lfloor 2.5K_L \rfloor]$.
- $K_C = 2(K_G - K_L)$ to allow some slack in selecting what variables can be chosen to satisfy the slowly varying constraint.

- The loss grid L_d^t is generated using two different methods:
  - **Uniform**: $L_d^t \sim U[0,1]$.
  - **Correlated**: $L_d^t \begin{cases} \sim U[0, D-d+1] & t \leq \lfloor T/2 \rfloor \\ \sim U[0,d] & t > \lfloor T/2 \rfloor \end{cases}$. This simulates a cost function where features with larger indices are more predictive for time periods $\leq \lfloor T/2 \rfloor$ and features with smaller indices are more predictive for time periods $\geq \lfloor T/2 \rfloor$.

For each T, D combination, we simulate the remaining parameters 100 times and record both the percentage of fully integral solutions and the percentage of integral variables obtained by the resulting linear optimization problem. The results are shown in Table 2.11 where Full Int. means fully integral solutions and Int. Var. represents percentage of variables that take integral values. We see that across all experiments and all types of cost grids, a significant portion of the solutions are integral. We further note that, in the case of non-integral solutions, the portion of non-integral entries in $\boldsymbol{z}$ is small (always less than 15% and, typically, even smaller).

Table 2.11: Results for integrality test of Algorithm 2.

D	T	Uniform		Correlated	
		Full Int. (%)	Int. Var (%)	Full Int. (%)	Int. Var (%)
2	2	100	100	100	100
2	5	69	92.2	63	92.8
2	10	46	91.7	51	91.0
5	2	100	100	100	100
5	5	37	88.6	47	92.4
5	10	8	85.1	9	86.5
10	2	100	100	100	100
10	5	42	95.1	47	95.7
10	10	24	93.7	35	94.1

Chapter 3

Variability and its Effect on the Stability of Decision Trees

3.1 Introduction

Machine learning (ML) algorithms get increasingly integrated into our lives [141]. Traditionally, the primary objective of ML has been to improve predictive power. The recent adoption of ML in high-stakes applications, ranging from health care [176, 74] to climate change [193, 80] has highlighted the need to achieve additional, often competing, objectives, including interpretability, robustness, and stability. Loosely defined, an interpretable ML model allows humans to have an understanding of the logic behind the model choices [171]. A robust ML model is protected against noise and different types of uncertainties in the data [229]. A stable ML model is not significantly affected by small changes in the data [65].

Decision tree models [63] are commonly used in high-stakes applications. Owing to their graphical visualization and discrete and sequential structure, which mimics the human thought process by successively asking questions that adapt based on previous answers, decision trees are inherently interpretable. In the words of Leo Breiman [66], "on interpretability, trees rate an A^+." The excellent interpretability of decision tree models, however, does not come for free: they are known to suffer in terms of all other objectives. "While trees rate an A^+ on interpretability, they are good, but not great,

predictors. Give them, say, a B on prediction." As we discuss in Section3.1.1, a lot of effort has been dedicated to materially improving decision trees' predictive power and robustness, while maintaining their interpretability [45]. Improving their stability has been addressed to a much lesser extent.

Example 1. *Consider a health care setting where we build an interpretable ML model to support a physician's decision making. A commonly encountered situation is that the initial dataset is small but larger amounts of data become available over time as more patients' information gets recorded. Then, it is reasonable to consider retraining the ML model to boost its predictive performance and potentially incorporate new patterns in the data. However, upon retraining, the new model often could change notably, owing to the instability of the underlying ML algorithm. This situation would harm interpretability and hinder the adoption of the model by the physician.*

The focus of this paper is the study of the stability of decision tree models through the lens of real-world health care applications. We introduce a quantitative (and, what we believe, practical) way to measure a decision tree's stability. Building on this measure of stability, we propose a methodology that enables training more stable decision trees. We investigate the potential existence of trade-offs between the different objectives in ML which we previously described. Ultimately, we hope that this work is a step towards addressing long-standing issues concerning the stability of decision tree models, and will further enhance their adoption in high-stakes applications.

3.1.1 Decision Tree Models and their Limitations

Improving predictive power and robustness. As we already discussed, the good-but-not-great predictive power of decision trees trained using heuristic methods such as CART [63] has been documented since early on after their inception. Recently, the use of mixed-integer optimization-based methods to learn globally optimal or near-optimal decision trees has led to notable improvements in predictive power — to the extent that such trees often compete with state-of-the-art black-box models [45, 46, 7, 9, 72]. Optimal decision trees exhibit improved stability compared to

heuristic ones, especially when used in combination with feature selection techniques [41], but still exhibit instability. Moreover, recent approaches have developed decision trees with additional desirable properties, including robustness to noise or adversarial perturbations in the data features [142, 169, 48] and fairness [6].

The source of instability. In a typical model selection procedure, a "best" ML model is chosen from a collection of predictors obtained, e.g., using different hyper-parameters. A procedure is called unstable if a small change in the data used to obtain the sequence of models can cause large changes in the best model [65]. This instability occurs commonly when there are many different models crowded together that have similar predictive power, in which case a slight change in the data can cause a change from one model to another. The two models are close to each other in terms of predictive power, but can be distant in terms of their structure. [66] describes this situation as the "Rashomon effect" and the set of all such models as the "Rashomon set;" the term is derived from Akira Kurosawa's 1950 film "Rashomon," in which a murder is described in four contradictory ways by four witnesses. [227] propose a technique for enumerating the Rashomon set for decision trees — without, however, any stability considerations.

Are stability and interpretability at odds? [230] show that, for sparse regression, stability and interpretability (owing to the sparsity of the model) are at odds with each other: sparser models are shown to be less stable. However, from a practical viewpoint, we observed in [44] that the stability of sparse regression models improves by enabling them to vary smoothly over, e.g., time and controlling the distance between the respective coefficients. In the context of decision trees, stabilization has been traditionally achieved at the expense of interpretability via the use of boosting [108, 79] or bagging [62, 64] — improving stability was, in fact, the primary motivation in developing Random Forest. Once again quoting [65], "While stable procedures have desirable properties, stabilization by averaging is not a panacea. An area that needs exploration is the possibility of stabilization of procedures by changing their structure

instead of averaging. An interesting research issue we are exploring is whether there is a more stable single-tree version of CART." In this paper, we hope to make progress toward answering this question.

3.1.2 Towards More Stable Decision Trees

Improving stability. Attempts to improve decision trees' stability have largely focused on (heuristically) tweaking either the learning algorithm or the model selection procedure. In the former case, [152] and [166] propose (slightly) more stable decision tree variants, such as directed graphs or trees with hyperplane splits, respectively, and [15] propose a series of tests to prevent internal instability in the tree-growing process. In the latter case, [197] define a probability distribution for an equivalence class of trees and select the maximum likelihood tree structure; [47] develop stable classification models by optimizing the choice of training/validation split, but their work does not materially improve the stability of decision trees. We propose a decision tree learning algorithm-agnostic **stabilization methodology**, which, by explicitly quantifying stability, allows us to identify the most stable trees in a collection thereof.

Measuring stability. The first step towards quantifying decision trees' stability comprises of measuring the distance between two trees. To do so, a first family of approaches [213, 67], referred to as "semantic stability," evaluates the degree to which two decision trees make the same predictions, e.g., by classifying a randomly selected set of instances and calculating the proportion assigned to the same class by both trees. The second family [239, 67], "structural stability," examines the similarity between structural properties of two trees, e.g., by looking at the variance in the size and depth of the trees during cross-validation. Hybrid approaches [94, 218] rely on region stability or compatibility and estimate the probability that the trees classify a randomly selected example in "equivalent" decision regions. None of the above approaches directly compares the two trees' structures; this would be possible using syntactic distance measures [153, 236, 187], such as the edit distance, which, however, heavily depend on the representation and consider logically equivalent trees

as different [213, 165]. We encode decision trees in a way that enables us to overcome this limitation and compute **trees' structural distance** in the most direct way — by finding the optimal matching of the trees' paths.

3.1.3 Interpretability and its Interface with Stability

Importance of interpretability in ML. The practical implications of interpretability on the adoption of ML models have been highlighted by numerous recent studies. Practitioners are more likely to use algorithms if they understand and are able to modify them [89]; this can be particularly beneficial when algorithmic decisions lack domain knowledge and suffer by model misspecification [77], or when human decision makers have access to private information that is unused by the algorithm [138, 19]. Vice versa, interpretable ML can assist and affect human decisions [118], or even help improve workers' performance by inferring tips and strategies from the model [22]. Especially in health care applications, the incorporation of ML into clinical medicine raises numerous ethical challenges [74] and it is crucial for practitioners to be able to understand the reasoning behind ML models' decisions. This raises questions regarding quantitatively assessing the effect of interpretability on model performance (see [49] for such a study in the context of algorithmic insurance).

Measuring interpretability. The preceding discussion emphasizes the need to further understand interpretable ML models. [38] introduce a mathematical framework to rigorously measure a model's interpretability, which, until recently, remained only loosely defined: using their framework, models that are in principle interpretable (including decision trees) can in practice have varying degrees of interpretability. Additionally, Example 1 raises the question of how interpretable can an unstable model be: a model which changes vastly when the data changes slightly cannot provide trustworthy knowledge concerning the underlying problem. In this paper, we explore the **stability-interpretability relationship** for decision tree models, which are inherently interpretable, and uncover the interpretability characteristics (e.g., number of nodes) of the most stable trees.

3.1.4 Contributions, Outline, and Methodology

Contributions. We now summarize the contributions of our work.

- We introduce a novel distance metric for decision tree models. The proposed metric enables us to quantify how structurally different two decision trees are and determine their relative stability.
- We propose a new methodology to train more stable decision tree models. The proposed methodology is particularly relevant in settings where more data is expected to become available over time, in which case the underlying ML model may need to be retrained.
- We demonstrate the value of the proposed methodology through a variety of real-world case studies in health care, where stability and interpretability are both essential, ranging from predicting the risk of deep vein thrombosis to examining the effect of radiotherapy on reducing the risk of local recurrence to patients with sarcoma tumor.

Outline. We organize the rest of the paper as follows. In Section 3.2, we formalize the decision tree problem and introduce the proposed distance metric for decision trees, which plays a central role in our notion of stability. Then, Section 3.3 presents and evaluates our methodology to train stable decision trees. Finally, in Section 3.4, we provide a detailed description and qualitative analysis of the six real-world case studies from the health care space which we use to empirically study the proposed methodology.

Experimental methodology and software. All numerical experiments in this paper rely on six real-world case studies, all of which come from applications in health care. In summary, the case studies we consider are: *Thrombosis*, where we predict the risk of deep vein thrombosis after endovenous thermal ablation; *Sarcoma tumor,* where we examine the effect of radiotherapy on reducing local recurrence within five years to patients with sarcoma tumor; *REBOA*, where we study whether, using ML,

we can decrease the misuse of resuscitative endovascular balloon occlusion of the aorta in hemodynamically unstable blunt trauma patients; *TAVR*, where we investigate whether using the appropriate valve type in a transcatheter aortic valve replacement procedure can reduce the need for pacemaker; *Splenic injury*, where we explore how different treatments affect mortality of victims of blunt splenic injury; and *Breast cancer*, where we predict whether a breast cancer is benign or malignant using features computed from a digitized image of a fine needle aspirate of a breast mass. We provide specific details about and a thorough qualitative analysis of the case studies in Section 3.4.

In all case studies, we split the data into training (67%) and testing (33%) sets, which we use to train and evaluate (out-of-sample), respectively, our models. We repeat the data spliting process multiple times (10, unless otherwise specified), and report the mean and standard deviation of the results. We implement all algorithms in `Python` programming language (version 3.7). We use the `Scikit-learn` implementation [183] of the CART [63] and Random Forest [62] algorithms. We solve the optimization models using the `Gurobi` commercial solver (version 9.5). All experiments were performed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running CentOS release 7.

3.2 Measuring the Distance between Decision Trees

Central to our approach is a distance metric between decision trees, which directly compares the trees' structures and, to our knowledge, has been missing from the ML literature; the subject of this section is the definition and study of such a metric.

3.2.1 Decision Tree Problem Definition

We start by introducing the decision tree problem and the notation we use throughout the paper.

Notation. We are given data $\boldsymbol{X} \in \mathbb{R}^{N \times P}$ and responses $\boldsymbol{y} \in \mathbb{R}^N$; noting that our work naturally generalizes to regression problems with $\boldsymbol{y} \in \mathbb{R}^N$, we focus, throughout this paper, on classification problems with classes $[K] := \{1, \ldots, K\}$ so that $\boldsymbol{y} \in [K]^N$. We denote by $\mathcal{N} \subseteq [P]$ the set of numerical features and $\mathcal{C} = [P] \setminus \mathcal{N}$ the set of categorical features. Each numerical feature is characterized by its upper bound $u_j, j \in \mathcal{N}$, and its lower bound $l_j, j \in \mathcal{N}$; each categorical feature is characterized by its number of categories $c_j, j \in \mathcal{C}$. Therefore, a problem's feature space is defined by a collection of three vectors $(\boldsymbol{u}, \boldsymbol{l}, \boldsymbol{c}) \in \mathbb{R}^{|\mathcal{N}|} \times \mathbb{R}^{|\mathcal{N}|} \times \mathbb{N}^{|\mathcal{C}|}$ corresponding, respectively, to the numerical features' upper and lower bounds, and the categorical features' number of categories.

Problem statement. In their simplest form, decision trees (with parallel splits) partition the feature space into a set of rectangles, and then assign a prediction to each; for classification problems, the assigned prediction is a class label $k \in [K]$. Each data point $\boldsymbol{x} \in \mathbb{R}^P$ is classified according to the class label that corresponds to the rectangle where it lies. The decision tree learning problem can be described as searching for a way to partition the feature space, such that an error metric, e.g., the number of misclassified points in the training data, is minimized — possibly subject to additional terms and constraints that, respectively, penalize or prevent more granular partitionings or, equivalently, more complex trees.

3.2.2 Decision Tree Representation

We now present a compact representation of a decision tree as a collection of paths, which facilitates the measurement of the distance between trees.

Definition of a split and feature space partitioning. A trained decision tree performs a sequence of "splits" on a subset of features. To split on a numerical feature $j \in \mathcal{N}$, we test whether its value is below a threshold $t \in [l_j, u_j]$. To split on a categorical feature $j \in \mathcal{C}$, we test its membership in a subset of categories $\mathcal{C}' \subseteq [c_j]$.

A sequence of splits partitions the feature space in the following way. Starting

at the root node, the tree performs a split on a feature and therefore partitions the feature space into two disjoint rectangles. In the rectangle where the condition is satisfied, t defines an upper bound for $j \in \mathcal{N}$ or $\mathcal{C}'$ determines the set of qualifying categories for $j \in \mathcal{C}$. In the rectangle where the condition is not satisfied, t defines a lower bound for $j \in \mathcal{N}$ or $(\mathcal{C}')^C$ determines the qualifying categories for $j \in \mathcal{C}$. Then, the tree (possibly) further partitions each of the two resulting rectangles, each by splitting on a (possibly different) feature. The same logic is applied to every node in the tree, until a leaf node is reached.

Definition of a tree path. After performing a full sequence of splits, we obtain one of the final rectangles. Each such sequence, together with the class label that is assigned to the resulting rectangle define a tree path. Path p is characterized by the upper and lower bounds that are imposed on numerical features, the qualifying categories for categorical features, and the assigned class label. We represent the qualifying categories for feature j in path p as binary vector $\boldsymbol{c}_j^p \in \{0,1\}^{c_j}$ with ones in positions that correspond to categories that qualify across path p. We then append zeros to all such vectors (so they are of the same length) and stack them to form matrix $\boldsymbol{C}^p \in \{0,1\}^{|\mathcal{C}| \times \max_j c_j}$. Put together, we represent a tree path as $(\boldsymbol{u}^p, \boldsymbol{l}^p, \boldsymbol{C}^p, k^p) \in \mathbb{R}^{|\mathcal{N}|} \times \mathbb{R}^{|\mathcal{N}|} \times \{0,1\}^{|\mathcal{C}| \times \max_j c_j} \times [K]$.

Representation of a tree. A tree $\mathbb{T}$ is then (non-uniquely) represented as a collection of T paths $\mathcal{P}(\mathbb{T}) = \{p_1, \ldots, p_T\}$, where $|\mathcal{P}(\mathbb{T})| = T$. The non-uniqueness of this representation of decision trees owes to the fact that the order in which splits are performed does not matter. Thus, multiple trees can result in the same collection of paths $\mathcal{P}$. We believe this is a desirable property, since two trees that result in the same collection of paths decide on which class label to assign to any data point by testing the exact same set of conditions (albeit in different order).

3.2.3 Distance between Paths

In this section, we define two quantities that serve as building blocks in measuring the distance between two trees: the distance between two paths and the notion of a path's weight.

Paths' distance. To measure the distance between two paths, we compare the feature ranges and the class label that each path results in. Intuitively, two paths are close if they result in overlapping rectangles. This leads to the following definition for the distance between paths p and q:

$$d(p,q) = \sum_{j \in \mathcal{N}} \frac{\left|u_j^p - u_j^q\right| + \left|l_j^p - l_j^q\right|}{2(u_j - l_j)} + \sum_{j \in \mathcal{C}} \frac{\left\|\boldsymbol{c}_j^p - \boldsymbol{c}_j^q\right\|_1}{c_j} + \lambda \cdot \mathbb{1}_{(k^p \neq k^q)}, \tag{3.1}$$

where $\mathbb{1}_{(\cdot)}$ denotes the indicator function. We weigh the last term in $d(p,q)$ by λ to adjust the relative importance in comparing the feature ranges and the class labels between the two paths. For example, two paths of depth D can result in different feature ranges for at most $2D$ features. By setting $\lambda = 2D$, we assign equal weight to the amount of overlap in feature ranges and the resulting class labels between the two paths. We remark that instead of comparing the paths' class labels, we can compare the leaf class distributions by simply replacing the indicator $\mathbb{1}_{(k^p \neq k^q)}$ in $d(p,q)$ with, e.g., the leaf node's Gini impurity. By doing so, the resulting paths' distance would incorporate statistical considerations too; as we are interested in structural stability, we do not address this here.

Path weight. In addition, to quantify a path's importance, we introduce the notion of "path weight," which captures the portion of the feature ranges that the path covers (among features that are used in the path's split nodes) and is defined as follows:

$$w(p) = \sum_{j \in \mathcal{N}} \frac{u_j^p - l_j^p}{u_j - l_j} \cdot \mathbb{1}_{\left(u_j^p \neq u_j \text{ or } l_j^p \neq l_j\right)} + \sum_{j \in \mathcal{C}} \frac{c_j^p}{c_j} \cdot \mathbb{1}_{\left(c_j^p \neq c_j\right)}. \tag{3.2}$$

Intuitively, a path that is assigned a heavy weight will lead to rectangles which include large portions of the ranges of the numerical features or many categories for the categorical features. The path weight will be used to measure the distance between trees with different numbers of paths.

Notice that, in measuring both the paths' distance and the path weight, each feature is divided by its range or number of categories, so both quantities are expressed in the same scale.

3.2.4 Distance between Trees and Computation

Using the decision tree representation of Section 3.2.2 and the path-related quantities of Section 3.2.3, we are ready to introduce the proposed distance metric for decision trees.

Trees' distance. To measure the distance between two trees, we look at how different their paths are in terms of the features each path splits on, the split thresholds or categories (for numerical or categorical features respectively), and the resulting class label. Such an approach captures structural differences between the two trees, instead of, e.g., comparing the distributions of outcomes. Intuitively, trees that are close will consist of similar paths and therefore lead to similar partitionings of the feature space. The proposed distance metric compares the two trees' paths and searches for a way to optimally match them.

Formally, we are interested in measuring the distance between trees $\mathbb{T}_1$ and $\mathbb{T}_2$. We assume, without loss of generality, that $T_1 > T_2$, that is, $\mathbb{T}_1$ consists of a larger number of paths. We introduce decision variables $x_{pq} = \mathbb{1}(\text{path } p \text{ in } \mathbb{T}_1 \text{ is matched with path } q \text{ in } \mathbb{T}_2)$ and $x_p = \mathbb{1}(\text{path } p \text{ in } \mathbb{T}_1 \text{ is left unmatched})$. We formulate the

following integer linear optimization problem:

$$
\begin{aligned}
d(\mathbb{T}_1, \mathbb{T}_2) \;=\; \min_{\boldsymbol{x}} \quad & \sum_{p \in \mathcal{P}(\mathbb{T}_1)} \sum_{q \in \mathcal{P}(\mathbb{T}_2)} d(p,q) x_{pq} + \sum_{p \in \mathcal{P}(\mathbb{T}_1)} w(p) x_p \\
\text{s.t.} \quad & \sum_{q \in \mathcal{P}(\mathbb{T}_2)} x_{pq} + x_p = 1, \quad \forall p \in \mathcal{P}(\mathbb{T}_1) \\
& \sum_{p \in \mathcal{P}(\mathbb{T}_1)} x_{pq} = 1, \quad \forall q \in \mathcal{P}(\mathbb{T}_2) \\
& x_{pq} \in \{0,1\}, \quad x_p \in \{0,1\}, \qquad \forall p \in \mathcal{P}(\mathbb{T}_1), \quad \forall q \in \mathcal{P}(\mathbb{T}_2)
\end{aligned}
\tag{3.3}
$$

Upon solving Problem (3.3), each path $p \in \mathcal{P}(\mathbb{T}_1)$ will be either matched with a path $q \in \mathcal{P}(\mathbb{T}_2)$, in which case the tree distance $d(\mathbb{T}_1, \mathbb{T}_2)$ will increase by the distance $d(p,q)$ between the two paths, or will remain unmatched, in which case the tree distance will increase by the path's weight $w(p)$. We note that, if $T_1 = T_2$, that is, the two trees consist of the same number of paths, we do not include x_p in the formulation.

The following lemma, which we prove in Appendix 3.6.1, suggests that the proposed distance measure satisfies the requirements of a metric: the distance from a tree to itself is zero, the distance between two distinct trees is positive, the distance is symmetric, and it satisfies the triangle inequality.

Lemma 8. *Let $\mathcal{T}$ denote the set of all trees of maximum depth D and $\mathbb{T}_1, \mathbb{T}_2 \in \mathcal{T}$. $d(\mathbb{T}_1, \mathbb{T}_2)$ is a metric mapping $\mathcal{T} \times \mathcal{T} \mapsto \mathbb{R}$.*

Computation. To compute the distance between two trees, our proposed approach requires solving Problem (3.3), which is a variant of bipartite matching and therefore efficiently solvable. In particular, consider the linear relaxation of Problem (3.3), where the binary constraints $x_{pq} \in \{0,1\}$ and $x_p \in \{0,1\}$ are replaced with linear constraints $0 \leq x_{pq} \leq 1$ and $0 \leq x_p \leq 1$. We have the following corollary, which, for completeness, we prove in Appendix 3.6.2:

Corollary 1. *Any extreme point of the linear relaxation of Problem* (3.3) *is a binary vector.*

Corollary 1 guarantees that the optimum to the linear relaxation of Problem 3.3 is

the incidence vector of a perfect matching and hence encodes an optimal matching of paths. Owing to this result, we can compute the distance between trees in polynomial time (and very efficiently in practice) by simply solving a linear optimization problem.

An upper bound on the distance. To get a relative (and more intuitive) sense of how close two trees are, we properly scale the distance so that it expresses a percentage of the maximal amount two trees of depth D can differ. To do so, we derive a problem-independent upper bound on the proposed distance metric for given maximum allowable tree depth D:

Lemma 9. *Given trees $\mathbb{T}_1$ and $\mathbb{T}_2$ with depth$(\mathbb{T}_1) \leq D$ and depth$(\mathbb{T}_2) \leq D$, it holds that $d(\mathbb{T}_1, \mathbb{T}_2) \;\leq\; 2^D\;(2D + \lambda)$.*

Owing to Lemma (9), which we prove in Appendix 3.6.3, upon computing the distance, we scale it by $1/2^D(2D+\lambda)$ and hence get an expression of the distance as a percentage of the distance between the two trees that are as far apart as possible.

3.2.5 Tree Distance in Practice

We now return to the setting described in Example 1 and study the proposed distance metric through a simple practical example. We use the sarcoma tumor case study (see Section 3.4 for details).

Using an initial dataset $\boldsymbol{X}_0 \in \mathbb{R}^{N/2 \times P}$ (which we build by sampling $\frac{N}{2}$ data points), we train two decision trees. For the first one, shown in the top-left panel of Figure 3-1 and referred to as `CART CV`$_0$, we apply a standard 5-fold cross-validation procedure. For the second one, shown in the bottom-left panel of Figure 3-1 and referred to as `CART Pareto`$_0$, we apply our proposed methodology (described in Section 3.3.1). At a later time, when the full dataset $\boldsymbol{X} \in \mathbb{R}^{N \times P}$ becomes available, we retrain the two decision trees by (independently) repeating the aforementioned two methodologies — using $\boldsymbol{X}$ instead of $\boldsymbol{X}_0$. We refer to the resulting trees as `CART CV` (top-right panel of Figure 3-1) and `CART Pareto` (bottom-right panel of Figure 3-1). We note that we

tuned all trees using the same set of candidate hyperparameters and, for simplicity in presentation, restricted the maximum depth to 4.

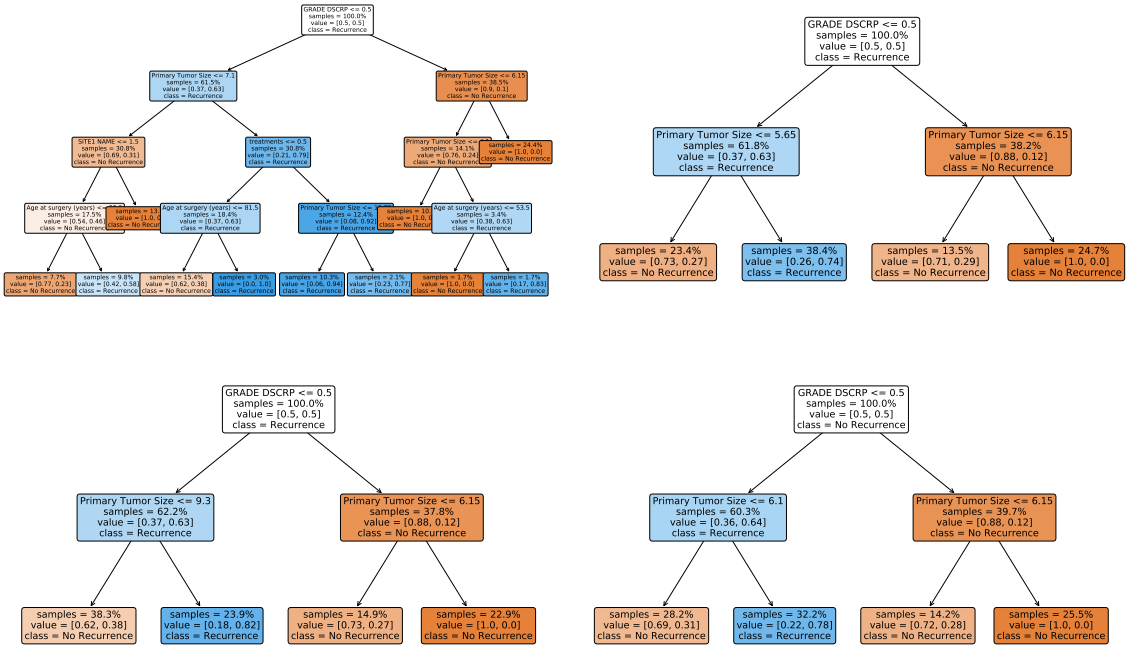


Figure 3-1: Trees obtained using cross-validation (top) versus trees obtained using the proposed stable methodology of Section 3.3.1 (bottom). For the left column trees, we used an initial training set of $\frac{N}{2}$ data points; for the right column trees, we used the full training set of N data points.

Clearly, the trees obtained using cross-validation are structurally much more different from each other compared to the trees obtained using the proposed stable methodology. $\texttt{CART CV}_0$ and $\texttt{CART CV}$ differ in both their depth (4 versus 2) and in the number of distinct features they split on (5 versus 2). In contrast, $\texttt{CART Pareto}_0$ and $\texttt{CART Pareto}$ split on the exact same features and the only difference is a slight shift in the left child node's threshold by about 7% of the corresponding feature range. The proposed distance metric captures this visual observation: $d(\texttt{CART CV}_0, \texttt{CART CV}) = 11.21\%$ and $d(\texttt{CART Pareto}_0, \texttt{CART Pareto}) = 0.18\%$, where the distances are expressed as a percentage of the distance between the two trees of depth 4 that are as far apart as possible. Closing our Example 1 analogy, we claim that a physician who had been using $\texttt{CART Pareto}_0$ will likely trust $\texttt{CART Pareto}$

upon retraining; it would be more difficult for them to transition from `CART CV`$_0$ to `CART CV`.

3.2.6 Sensitivity Analysis of the Proposed Tree Distance

Before concluding the section, we discuss some key numerical properties of the proposed tree distance metric. The corresponding numerical study, which uses the case studies of Section 3.4, is given in Appendix 3.7. We test the sensitivity of the proposed distance metric to two types of perturbations — direct and indirect ones. Direct perturbations refer to immediate interventions and changes in the tree structure. Indirect perturbations refer to modifications in the training data. The main takeaways are the following:

- The proposed distance metric has desirable properties, including having a linearly increasing relationship with the amount of direct perturbation in the tree and showing an increasing trend with the amount of indirect perturbation in the tree (i.e., the fraction of the training data that changes).

- The upper bound derived in Section 3.2.4 is conservative in that, trees where the thresholds are allowed to vary by 50% in expectation and 100% at maximum have an expected distance of 12.5% and a standard deviation in the distance of 5%, expressed as percentage of the maximum possible distance between any two trees of the given depth.

3.3 Computing Stable Decision Trees

In this section, we develop and empirically evaluate a methodology that improves the stability of decision tree models, using the distance metric we introduced in Section 3.2.

3.3.1 Training Collections of Stable Trees

We now describe the proposed methodology to train collections of stable decision trees. The methodology is motivated by the health care setting of Example 1. We have an initial patient database (corresponding to, e.g., any of the case studies we describe in Section 3.4) and train a model using the data available at the time. Later, when more patient data becomes available, we retrain the model to improve its accuracy (owing to the larger sample size) and to capture new patterns that may be present in the data. It is crucial that the new model does not deviate too much from the previous one, as this could affect physicians' trust and willingness to use the model. We address such concerns by incorporating a notion of stability in the tree training and selection process.

More concretely, the proposed methodology consists of the following steps:

- We randomly split the training data $\boldsymbol{X}$ into two batches, $\boldsymbol{X}_0 \in \mathbb{R}^{N_0 \times P}$ and $\boldsymbol{X}_1 \in \mathbb{R}^{N_1 \times P}$, such that $N_0 + N_1 = N$.

- Using the first batch of training data, $\boldsymbol{X}_0$, we train a first collection of B trees $\mathcal{T}_0 = \{\mathbb{T}_1^0, \ldots, \mathbb{T}_B^0\}$. We obtain $\mathcal{T}_0$ by bootstrapping $\boldsymbol{X}_0$, i.e., generating multiple new training datasets by sampling $\boldsymbol{X}_0$ uniformly and with replacement. For each dataset, we train decision trees with different hyperparameters: tree depth $D \in \{3, \ldots, 12\}$ and minimum number of samples per leaf $M \in \{3, 5, 10, 30, 50\}$.

- Using the full training data, $\boldsymbol{X}$ (hence merging the two batches $\boldsymbol{X}_0$ and $\boldsymbol{X}_1$), we train a second collection of B trees $\mathcal{T} = \{\mathbb{T}_1, \ldots, \mathbb{T}_B\}$, in the exact same way: we bootstrap $\boldsymbol{X}$ and examine, for each dataset, a large set of hyperparameters.

- We then compute, for every tree in the second batch $\mathbb{T}_b \in \mathcal{T}, b \in [B]$, its mean distance from all trees in the first batch

$$d_b = \sum_{\beta=1}^{B} \frac{d(\mathbb{T}_\beta^0, \mathbb{T}_b)}{B}.$$

  In addition, using holdout (validation or testing) data $(\boldsymbol{X}_{\text{holdout}}, \boldsymbol{y}_{\text{holdout}})$, we

estimate, for every tree in the second batch $\mathbb{T}_b \in \mathcal{T}, b \in [B]$, its out-of-sample predictive performance. For example, in binary classification problems, we use the area under the ROC curve

$$\alpha_b = \text{AUC}(\mathbb{T}_b; \boldsymbol{X}_{\text{holdout}}, \boldsymbol{y}_{\text{holdout}}).$$

- We obtain the collection $\{(\mathbb{T}_b, d_b, \alpha_b)\}_{b=1}^{B}$, where each tree $\mathbb{T}_b \in \mathcal{T}$ is characterized by two, possibly competing, metrics: d_b, characterizing its stability, and α_b, characterizing its predictive power. For any tree $\mathbb{T}_b$, we need to guarantee that there exists no another tree that performs at least as well on one metric and strictly better on the other; if such a tree existed, it would be strictly preferred for all practical considerations. To address this situation, we search for the Pareto frontier, that is, the set of Pareto optimal trees. A tree $\mathbb{T}_b \in \mathcal{T}$ is Pareto optimal if there exists no other tree $\mathbb{T}_{b'} \in \mathcal{T}, b \neq b'$, with

  $$(d_{b'} \leq d_b \text{ and } \alpha_{b'} > \alpha_b) \text{ or } (d_{b'} < d_b \text{ and } \alpha_{b'} \geq \alpha_b).$$

  We denote the set of Pareto optimal trees (satisfying the aforementioned condition) by $\mathcal{T}^\star \subseteq \mathcal{T}$.

- Once $\mathcal{T}^\star$ is computed, we can select a tree from the Pareto frontier using an application-specific function:

  $$\mathbb{T}^\star = \underset{\mathbb{T}_b \in \mathcal{T}^\star}{\text{argmax}}\ f(d_b, \alpha_b). \tag{3.4}$$

  For example, for a given suboptimality tolerance ϵ, we can use: $f(d_b, \alpha_b) = (1 - d_b) \cdot \mathbb{1}\left(\alpha_b \geq (1-\epsilon)\max_{b'} \alpha_{b'}\right)$. Alternatively, if we are equally interested in stability and predictive power, we can choose the tree $\mathbb{T}^\star$ that maximizes $f(d_b, \alpha_b) = \frac{-d_b + \alpha_b}{2}$.

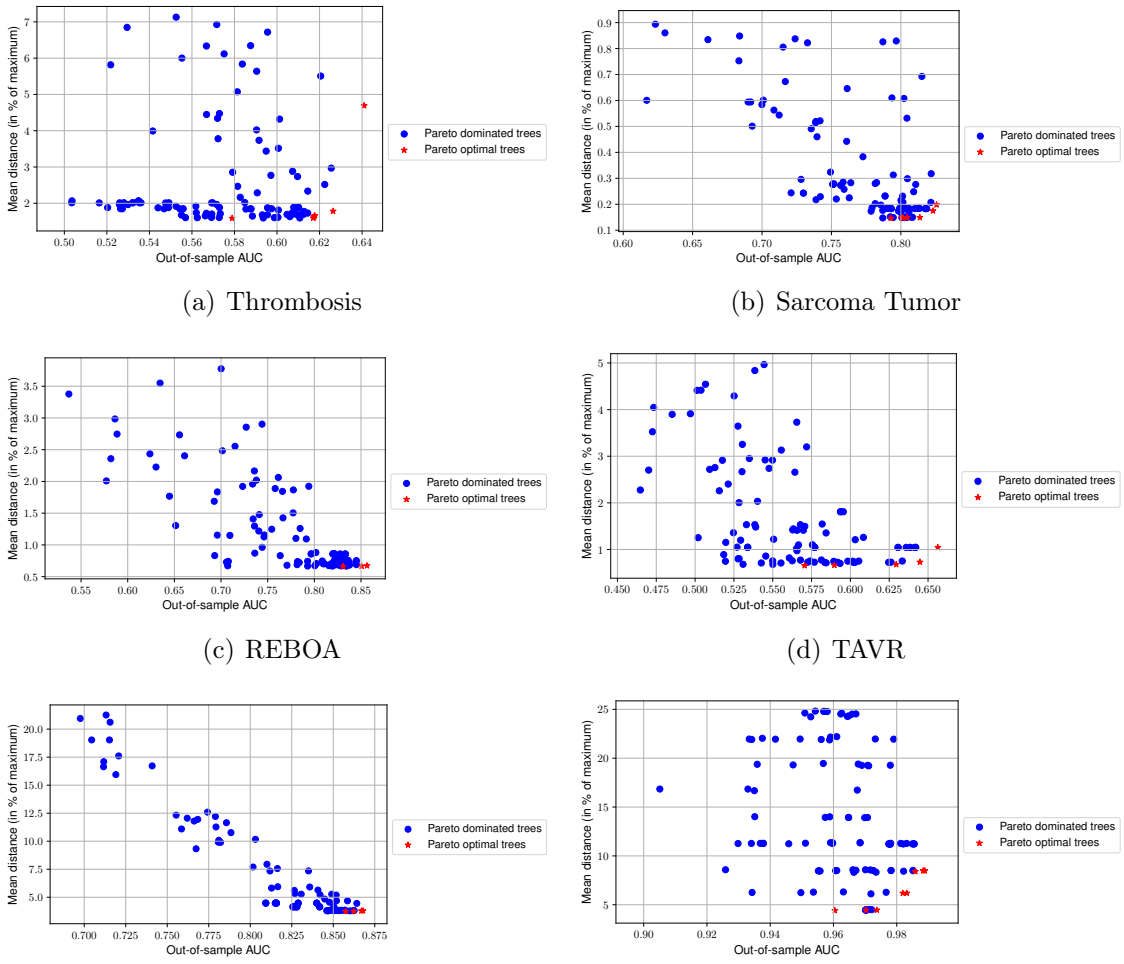
The proposed methodology relies on the assumption that decision trees' stability is negatively correlated with the proposed tree distance metric. By selecting the tree in

the second batch that is closest, on average, to the first batch trees, we are choosing a "centroid" tree, which is similar to many among a large number of trees obtained using a large number of datasets and parameters. Intuitively, such tree is likely to be a stable one.

As a final remark, we note that it is possible to characterize the trained trees in terms of additional metrics. For example, we may be willing to assign an interpretability score i_b (encoding, e.g., the number of nodes in the tree) to each tree. In that case, we would need to identify Pareto optimal trees in three dimensions and extract the set $\mathcal{T}^\star$ using the collection $\{(\mathbb{T}_b, d_b, \alpha_b, i_b)\}_{b=1}^B$. The output tree would then be selected using a function $f(d_b, \alpha_b, i_b)$ of all metrics of interest.

3.3.2 Pareto Optimal Trees

We now numerically test our hypothesis on the existence of a set of Pareto optimal trees using the case studies of Section 3.4. For each case study, we train a collection of trees $\mathcal{T}$ using the methodology of Section 3.3.1 and, for each tree $\mathbb{T}_b$, we calculate its out-of-sample AUC α_b and mean distance from the first batch d_b. We plot the results in Figure 3-2. In red, we show the set $\mathcal{T}^\star$ of Pareto optimal trees; in blue, we show the set of Pareto dominated trees. The formation of a Pareto frontier is clearer in the Thrombosis, Sarcoma tumor, TAVR, and Breast cancer case studies; in the remaining case studies, the set of Pareto optimal trees is concentrated in the bottom right corner of the graph — in which case the stability-predictive power trade-off is less profound, and the selection of the final tree should be easier. In Figure 3-3, we give summary statistics about the size of the Pareto frontier for each case study and across 10 independent repetitions of the experiment (training-testing data splits). At maximum, the frontier never exceeds nine trees; on average, it consists of only five trees; in one case, there exists just one Pareto optimal tree.



(a) Thrombosis

(b) Sarcoma Tumor

(c) REBOA

(d) TAVR

(e) Splenic injury

(f) Breast cancer

Figure 3-2: Trade-off between stability and predictive performance for collection of trees.

3.3.3 The Effect of Stability on Predictive Performance

This section aims to further understand the trade-off between stability and predictive performance. In Figure 3-4(left), we benchmark, for each case study and in terms of their out-of-sample AUC, two Pareto optimal trees (the AUC-maximizing, referred to as "`CART Pareto AUC`", and the distance-minimizing, referred to as "`CART Pareto Distance`") against the best tree obtained using a standard 5-fold cross-validation procedure (referred to as "`CART CV`") and against random forest ("`RF`''). Random forest is known to significantly improve upon the stability and predictive performance of decision trees [62], at the expense of interpretability. We make the following observations:

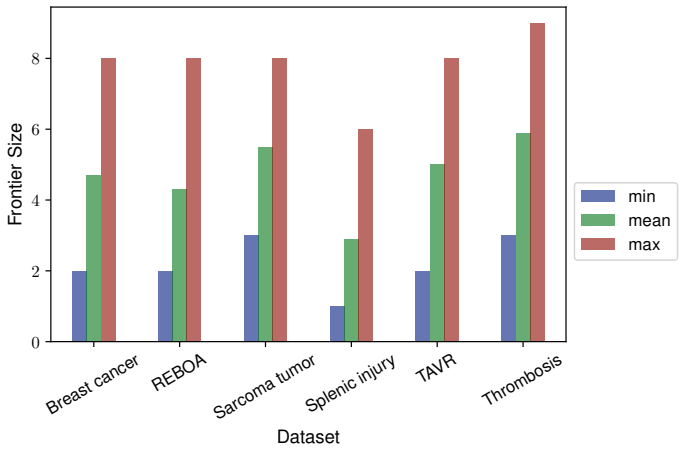


Figure 3-3: Minimum, mean, and maximum number of Pareto optimal trees for each dataset across 10 data splits.

- In all case studies, `CART Pareto AUC` significantly outperforms `CART Pareto Distance` and `CART CV`. In the Sarcoma tumor and TAVR case studies, `CART Pareto AUC` outperforms `RF`, whereas in Breast cancer it competes closely.

- `CART Pareto Distance` outperforms `CART CV` in three case studies in terms of mean AUC and in four case studies in terms of standard deviation. This leads to the conclusion that `CART Pareto Distance` strictly dominates `CART Pareto CV`; thus, in the sequel, we compare the two Pareto optimal trees against `RF`.

In addition, for each case study, we report aggregated results on the mean distance of `CART Pareto AUC` and `CART Pareto Distance` from the trees in the first batch (Figure 3-4(right)). We see that, except for the Breast cancer case study, the mean distance, for both trees, never exceeds 5% of the maximum possible distance, while the standard deviation of the distance (obtained over multiple repetitions of each experiment) is also small. This suggests that the proposed methodology enforces a significant level of stability.

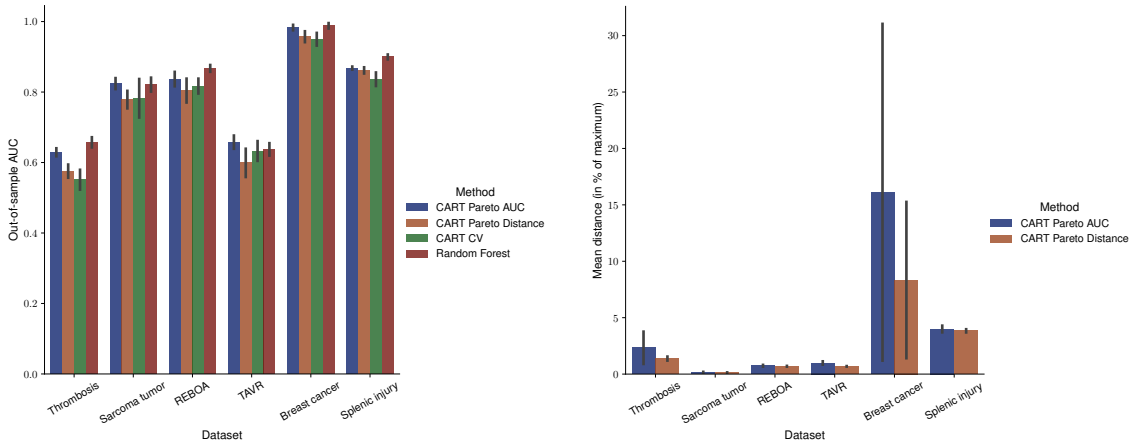


Figure 3-4: Mean and standard deviation of out-of-sample AUC (left) and stability (right) for each dataset across 10 data splits. In the left figure, we compare the AUC-maximizing Pareto optimal tree, the distance-minimizing Pareto optimal tree, the tree obtained using cross-validation, and random forest. In the right figure, we study stability in terms of the mean distance between the AUC-maximizing/distance-minimizing Pareto optimal tree and the trees obtained using the first batch of training data.

3.3.4 The Effect of Stability on Interpretability

We now investigate what implications our notion of stability has on the interpretability of the selected trees. To quantify interpretability, we compare the tree depth (Figure 3-5(left)) and number of nodes in the tree (Figure 3-5(right)) between `CART Pareto AUC`, `CART Pareto Distance`, and the largest tree in `RF`. In five out of six case studies, the proposed methodology results in much simpler trees compared to `RF`, which, on average, are half as deep and consist of two to ten times fewer nodes. Moreover, in four out of six case studies, the `CART Pareto AUC` tree is deeper and consists of more nodes compared to the `CART Pareto Distance` tree. This suggests that the proposed notion of stability is more likely to result in simpler and hence more interpretable trees.

3.3.5 The Effect of Stability on Feature Importance

We now study the effect of the proposed notion of stability on feature importances (or, equivalently, relevances), measured through the Gini importance [130, 164]. The

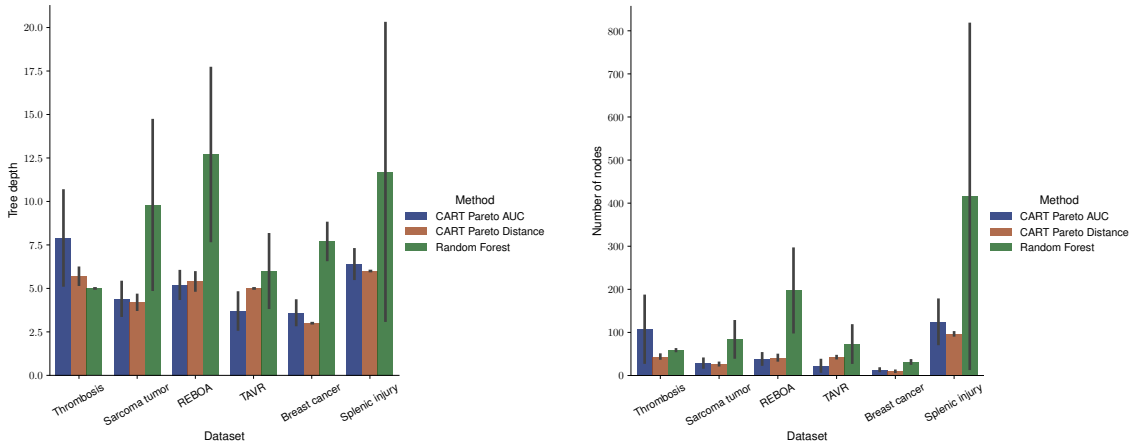


Figure 3-5: Mean and standard deviation of tree depth (left) and number of nodes (right) for each dataset across 10 data splits. We compare the AUC-maximizing Pareto optimal tree, the distance-minimizing Pareto optimal tree, and the largest tree in random forest.

Gini importance is a commonly used feature importance proxy that relies on the Gini impurity, which, in turn, measures the homogeneity of the target variable within different nodes in the tree. More concretely, the Gini impurity is a measure of how often a randomly chosen training data point from a node would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the node. The Gini importance of a feature is computed as the (normalized) total reduction of the Gini impurity brought by that feature, by measuring how often the feature was selected for a split and how large its overall discriminative value was.

We compare the standard deviation of feature importances averaged across all features (Figure 3-6(left)) and the total number of distinct features ranked as top-3 based on their feature importances (Figure 3-6(right)) between `CART Pareto AUC`, `CART Pareto Distance`, and `RF`. `RF` leads to smaller standard deviation and fewer distinct features marked as important; between `CART Pareto AUC` and `CART Pareto Distance`, the latter achieves smaller standard deviation in four out of six studies and marks more features as important in only one out of six studies hence suggesting that the proposed stability notion is positively correlated with having small deviations in feature importance.

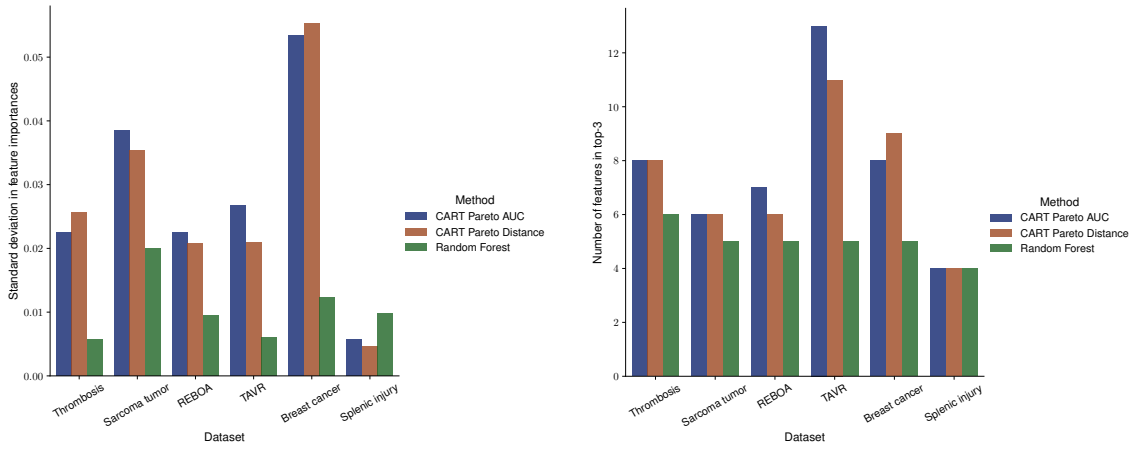


Figure 3-6: Standard deviation of feature importances, averaged across all features (left), and total number of distinct features ranked as top-3 based on their feature importances (right) for each dataset across 10 data splits.

3.3.6 Main Takeaways

We close this section by providing, in Table 3.1, a summary comparison between the AUC-maximizing and the distance-minimizing Pareto optimal trees across all experiments. The price of a 38% improvement in stability is, on average, 0.04 (or 4.625%) in AUC. The distance-minimizing Pareto optimal tree reduces the standard deviation in feature importances by 3.6%, the number of distinct important features by 4.4%, the number of nodes in the tree by 22.2%, and the tree depth by 6.2%.

Table 3.1: Aggregated comparison between the two extreme Pareto optimal trees.

Method	AUC	Distance	Feat. Import. Std.	Feat. in Top-3	Nodes	Tree Depth
`CART Pareto AUC`	0.8 (0.013)	4.056 (2.818)	0.028	7.667	55.867	5.2
`CART Pareto Distance`	0.763 (0.024)	2.511 (1.219)	0.027	7.333	43.5	4.883

3.4 Case Studies

In this section, we describe in detail the six real-world case studies we have used throughout the paper (Section 3.4.1) and analyze the trees trained using the proposed framework (Section 3.4.2).

3.4.1 Description of the Case Studies

All case studies come from the health care space due to the relevance of interpretable ML in this space. In the first five case studies, we obtain the data from collaborations with major US hospitals, including the Massachusetts General Hospital (MGH), the Hartford Hospital (HH), and the Memorial Sloan Kettering Cancer Center (MSK); for reproducibility purposes, in the last case study ("breast cancer"), we use a publicly available dataset. Table 3.2 provides a description of the datasets we use, including the origin of each dataset, the number of samples and features, and the outcome prevalence (i.e., the proportion of samples with the health outcome under consideration).

Table 3.2: Summary of datasets we use in our case studies.

Dataset	Origin	Number of Samples	Number or Features	Outcome Prevalence (in %)
Thrombosis	MGH Trauma Department	21,549	35	1.59
Sarcoma tumor	MSK	930	17	16.56
REBOA	MGH Trauma Department	10,000	30	31.27
TAVR	HH Cardiovascular Department	2,148	42	15.18
Splenic injury	MGH Trauma Department	35,954	41	6.1
Breast cancer	UCI	569	30	37.26

Thrombosis. We are interested in predicting the risk of deep vein thrombosis (DVT) after endovenous thermal ablation. The features include demographic factors (e.g., age, sex, ethnicity) as well as categorical (e.g., cigarette smoking history, wound infection, baseline dyspnea) and continuous (e.g., preoperative measurements such as creatinine, hematocrit, platelet) risk factors.

DVT is the most common cause of chronic venous disease, a widespread disease with an annual incidence of 1-2% and prevalence up to 73% in women and 56% in men. Endovenous treatments, such as thermal and laser ablation, are safe and effective, offering better outcomes and lower complications compared to traditional surgery. Yet, there are still risks to these procedures, including the development of a DVT, with a complication rate of around 1%, due to the subsequent risk of a pulmonary embolism. The risk of developing DVT ranges substantially in each individual patient due to specific risk factors, which motivates the study and development of prediction tools that estimate the risk of DVT after endovenous ablation [161].

Sarcoma tumor. We examine the effect of radiotherapy on reducing local recurrence within five years to patients with sarcoma tumor. The features include demographic factors (e.g., age at surgery), categorical (e.g., radiosensitivity) and continuous (e.g., primary tumor size) risk factors, and treatment-related factors.

Sarcomas are rare cancers that develop in the bones and soft tissues, including fat, muscles, blood vessels, nerves, deep skin tissues, and fibrous tissues. According to the National Cancer Institute, about 12,000 cases of soft tissue sarcomas and 3,000 cases of bone sarcomas are diagnosed in the U.S. each year. Sarcoma is treated with a combination of chemotherapy, radiation therapy, and surgery. Patients who have received radiation therapy for previous cancers may have a higher risk of developing a sarcoma. E.g., after treatment of primary soft tissue sarcomas, 11% to 14% of patients develop local recurrence [97], which may require additional surgery, radiotherapy, or even amputation, and may predict decreased overall survival. Most local recurrences arise in the first 5 years after diagnosis [112].

REBOA. We study whether, using ML, we can decrease the misuse of resuscitative endovascular balloon occlusion of the aorta (REBOA) in hemodynamically unstable blunt trauma patients. The features include demographic factors, categorical (e.g., Glasgow Coma Scale) and continuous (e.g., pulse and respiratory rates) risk factors, and treatment-related factors.

REBOA is a procedure that involves placement of an endovascular balloon in the aorta to control bleeding, augment afterload, and maintain blood pressure temporarily in traumatic hemorrhagic shock. The balloon blocks the artery and temporarily stops the blood flow giving doctors time to operate, but maintains blood circulation in the brain and heart. However, the parts of the body below the balloon are cut off from the normal blood flow, which may result in short- or longer-term problems [179, 140].

TAVR. We investigate whether using the appropriate valve type in a transcatheter aortic valve replacement (TAVR) procedure can reduce the need for pacemaker. The features include demographic factors, categorical (e.g., diabetes) and continuous

(e.g., left ventricular ejection fraction) risk factors, and treatment-related factors (manufacturer and type of the valve). TAVR is a minimally invasive heart procedure to replace a thickened aortic valve that cannot fully open, in which case blood flow from the heart to the body is reduced. TAVR can help restore blood flow and may be an option for people who are at risk of complications from surgical aortic valve replacement: in those patients, TAVR significantly reduces the rates of death and cardiac symptoms [71]. Nevertheless, there are multiple risks associated with TAVR, e.g., problems with the replacement valve (e.g., the valve slipping out of place or leaking), arrhythmias, and the need for a pacemaker.

Splenic injury. We explore how different treatments affect mortality of victims of blunt splenic injury. The features include demographic factors, categorical (e.g., splenic injury grade) and continuous (e.g., pulse and respiratory rate) risk factors, and treatment-related factors (splenectomy, angioembolization, or observation).

Blunt splenic injury occurs when a significant impact from some outside source (e.g., automobile accident) damages or ruptures the spleen. The traditional treatment has been splenectomy, the surgical procedure that partially or completely removes the spleen. The spleen is an important organ in regard to immunological function due to its ability to efficiently destroy encapsulated bacteria; its removal runs the risk of overwhelming post-splenectomy infection, a medical emergency and rapidly fatal disease caused by the inability of the body's immune system to properly fight infection following splenectomy [207]. Therefore, whenever possible, splenectomy is avoided to prevent the resulting permanent susceptibility to bacterial infections: most small, and some moderate-sized lacerations in stable patients are managed with hospital observation and sometimes transfusion rather than surgery; angioembolization, blocking off of the hemorrhaging vessels, is a less invasive treatment [209].

Breast cancer. We predict whether a breast cancer is benign or malignant using features computed from a digitized image of a fine needle aspirate of a breast mass. The features describe characteristics of the cell nuclei present in the image and in-

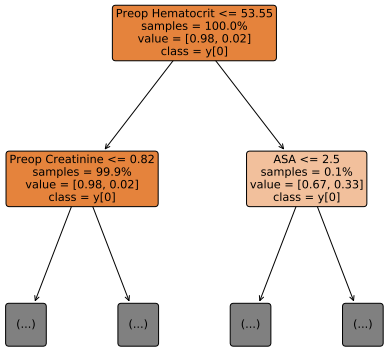
clude, specifically, for each nucleus, information about its radius, texture, perimeter, area, smoothness, compactness, concavity, number of concave points, summetry, and fractal dimension [222, 159]. The data is publicly available at the UCI ML repository at `https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)`.

3.4.2 Qualitative Analysis of the Selected Trees

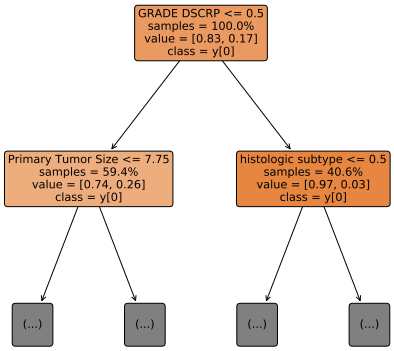
We now take a qualitative approach to analyzing the stability of the trees trained using the proposed framework. For each case study, we choose the tree from $\mathcal{T}^\star$ that maximizes Equation (3.4). We show, for each such tree, the first two levels of splits (at the root node, its left, and its right child) in Figure 3-7. More specifically, for each split node, we present the split feature and threshold, the proportion of total samples, the proportion of samples from each class, and the class label at that node. We compare the selected trees with the results given in Table 3.3, where, for each case study and among all trees in the full collection $\mathcal{T}$, we record the two most commonly selected features in each of the first three splits, along with their selection frequencies.

Table 3.3: Two most commonly selected features (and their corresponding selection frequencies) in each of the first two levels of splits, among all trained trees and across all repetitions for each case study.

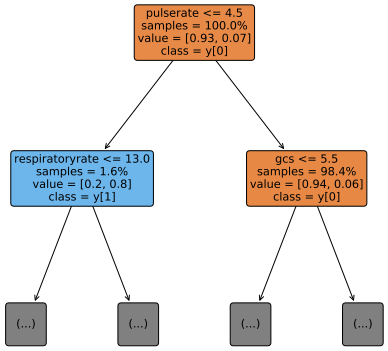
Dataset	**Split**	**Feature 1**	**Frequency (in %)**	**Feature 2**	**Frequency (in %)**
Thrombosis	Root	Preop Hematocrit	59.76	Preop Platelet	14.33
	Left	Preop Platelet	18.86	Age	16.48
	Right	*Leaf node*	60.48	Preop Hematocrit	6.38
Sarcoma tumor	Root	GRADE DSCRP	47.33	Chemo 2	25.00
	Left	Primary Tumor Size	59.00	Chemo 2	12.33
	Right	GRADE DSCRP	30.33	margin width	29.10
REBOA	Root	pulserate	81.00	gcs	10.00
	Left	age	24.10	respiratoryrate	18.71
	Right	gcs	90.00	respiratoryrate	5.33
TAVR	Root	ConductionDefect	63.00	VALVETYPE	14.00
	Left	VALVETYPE	44.00	ConductionDefect	13.00
	Right	LVEF	19.67	Area-Oversize	16.00
Splenic injury	Root	totalgcs	100.00	-	-
	Left	tbi	70.00	age	30.00
	Right	age	100.00	-	-
Breast cancer	Root	worst perimeter	38.52	mean concave points	22.00
	Left	worst concave points	35.38	worst area	12.00
	Right	worst texture	17.38	*Leaf node*	13.00



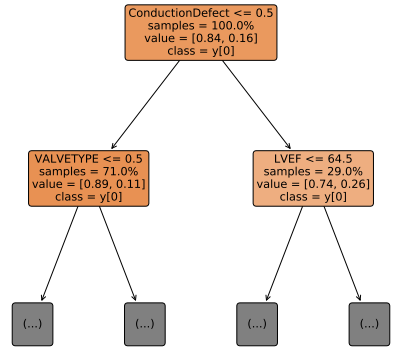
(a) Thrombosis



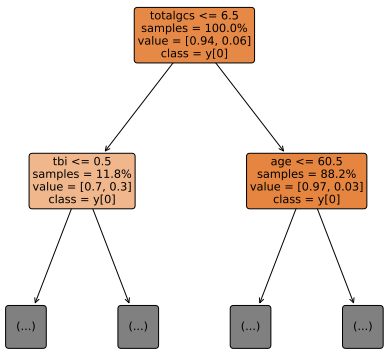
(b) Sarcoma Tumor



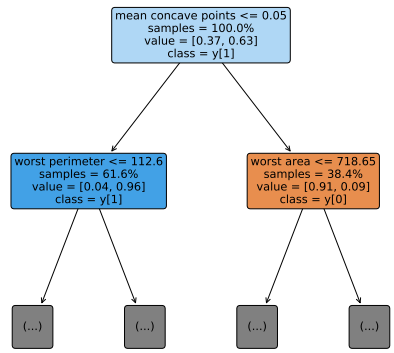
(c) REBOA



(d) TAVR



(e) Splenic injury



(f) Breast cancer

Figure 3-7: Visualization of the first two levels of splits in the Pareto optimal tree obtained using Equation (3.4) for each case study.

In summary, the selected trees are representative of the aggregate statistics. In *TAVR* (Figure 3-7(middle right)) and *Splenic injury* (Figure 3-7(bottom left)), the trees perform all splits in the first two levels on the most commonly selected features across all trained trees, suggesting that the selected trees are indeed stable. For *TAVR*, these features are the existence of a conduction defect (the root split is performed on this feature in 63% of the trees), the type of valve used (left split in 44%), and the left ventricular ejection fraction (right split in 20%). For *Splenic injury*, these features are the Glasgow Coma Scale (abbreviated "totalgcs", root split in 100%), the existence of traumatic brain injury (abbreviated "tbi", left split in 70%), and the age (right split in 100%). *REBOA* (Figure 3-7(middle left)), the tree splits on the pulse rate (root split in 81%) and then on the respiratory rate (second most common left split feature in 19%) and the Glasgow Coma Scale (abbreviated "gcs", right split in 90%). In *Sarcoma tumor* (Figure 3-7(top right)), the tree splits on the sarcoma grade (suggesting how abnormal the cancer cells look under a microscope — root split in 47%) and then on the primary tumor size (left split in 59%) and the histologic subtype (which is not among the most commonly selected features). In *Thrombosis* (Figure 3-7(top left)), the tree splits on the preoperative hematocrit levels (root split in 60%) and then on the preoperative creatinine levels and the ASA score, a metric to determine if someone is healthy enough to tolerate surgery and anesthesia (which are not among the most commonly selected features). Finally, in *Breast cancer* (Figure 3-7(bottom right)), the tree splits on the mean concave points feature (second most common root split feature in 22%) and then on the worst perimeter (most common root split feature in 39%) and the worst area (second most common left split feature in 12%).

3.5 Conclusion

In this paper, we have developed a methodology to improve the stability of decision tree models. The proposed methodology enables us to investigate trade-offs that are inherent to decision tree models and train stable, "Pareto optimal" decision trees; we demonstrate the value of the proposed approach through six extensive quantitative and

qualitative case studies from the health care space, where interpretability is essential. Further, we introduce a novel distance metric for decision trees, which has been missing from the ML literature, and use it to determine a tree's level of stability; the proposed distance metric may be of independent interest and can be used in different contexts, including, e.g., as a new way to impose regularization in decision tree models.

We conclude by returning to Leo Breiman's question, "is there a more stable single-tree version of CART?" Our work suggests that such a tree may indeed exist and, although it is unlikely to achieve the levels of stability of procedures that rely on averaging, the proposed Pareto optimal trees achieve improved predictive power (see, e.g., Section 3.3.3), interpretability (see Section 3.3.4), and are structurally stable (see Section 3.4.2).

3.6 Technical Proofs

3.6.1 Proof of Lemma 8

Recall that we represent a path p as a collection of two vectors (upper and lower bounds for numerical features), a matrix (categories for categorical features), and a scalar (class label): $(\boldsymbol{u}^p, \boldsymbol{l}^p, \boldsymbol{C}^p, k^p) \in \mathbb{R}^{|\mathcal{N}|} \times \mathbb{R}^{|\mathcal{N}|} \times \{0,1\}^{|\mathcal{C}| \times \max_j c_j} \times [K] \ := \ \mathcal{P}$. Without loss of generality, we flatten matrix $\boldsymbol{C}^p$ into a vector of appropriate dimension. We first prove the following auxiliary lemma:

Lemma 10. *Let $\mathcal{P}$ denote the set of all paths of depth D and $p, q \in \mathcal{P}$. Then*

$$d(p,q) = \sum_{j \in \mathcal{N}} \frac{\left|u_j^p - u_j^q\right| + \left|l_j^p - l_j^q\right|}{2(u_j - l_j)} + \sum_{j \in \mathcal{C}} \frac{\left\|\boldsymbol{c}_j^p - \boldsymbol{c}_j^q\right\|_1}{c_j} + \lambda \cdot \mathbb{1}_{(k^p \neq k^q)}$$

is a metric mapping $\mathcal{P} \times \mathcal{P} \mapsto \mathbb{R}$.

Proof of Lemma 10. We need to show that **(i)** $d(p,p) = 0$, **(ii)** if $p \neq q$, then $d(p,q) > 0$, **(iii)** $d(p,q) = d(q,p)$, **(iv)** $d(p,q) \leq d(p,r) + d(r,q)$. Axioms **(i)**-**(iii)** are trivially satisfied. We now prove axiom **(iv)**. Denoting $[\boldsymbol{\mu}]_j = \frac{1}{2(u_j - l_j)} > 0$, $[\boldsymbol{\nu}]_j = \frac{1}{c_j} > 0$, and by $\boldsymbol{x} \odot \boldsymbol{y}$ the element-wise product between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we

have:

$$\begin{aligned} d(p,q) &= \|\boldsymbol{\mu} \odot (\boldsymbol{u}^p - \boldsymbol{u}^q)\|_1 + \|\boldsymbol{\mu} \odot (\boldsymbol{l}^p - \boldsymbol{l}^q)\|_1 + \|\boldsymbol{\nu} \odot (\boldsymbol{C}^p - \boldsymbol{C}^q)\|_1 + \lambda \cdot \mathbb{1}_{(k^p \neq k^q)} \\ d(p,r) &= \|\boldsymbol{\mu} \odot (\boldsymbol{u}^p - \boldsymbol{u}^r)\|_1 + \|\boldsymbol{\mu} \odot (\boldsymbol{l}^p - \boldsymbol{l}^r)\|_1 + \|\boldsymbol{\nu} \odot (\boldsymbol{C}^p - \boldsymbol{C}^r)\|_1 + \lambda \cdot \mathbb{1}_{(k^p = k^r)} \\ d(r,q) &= \|\boldsymbol{\mu} \odot (\boldsymbol{u}^r - \boldsymbol{u}^q)\|_1 + \|\boldsymbol{\mu} \odot (\boldsymbol{l}^r - \boldsymbol{l}^q)\|_1 + \|\boldsymbol{\nu} \odot (\boldsymbol{C}^r - \boldsymbol{C}^q)\|_1 + \lambda \cdot \mathbb{1}_{(k^r = k^q)} \end{aligned}$$

We apply the triangle inequality to the first summand:

$$\begin{aligned} \|\boldsymbol{\mu} \odot (\boldsymbol{u}^p - \boldsymbol{u}^q)\|_1 &= \sum_{j \in \mathcal{N}} |\mu_j (u_j^p - u_j^q)| \\ &= \sum_{j \in \mathcal{N}} \mu_j |u_j^p - u_j^q| \\ &\leq \sum_{j \in \mathcal{N}} \mu_j (|u_j^p - u_j^r| + |u_j^r - u_j^q|) \\ &= \sum_{j \in \mathcal{N}} |\mu_j (u_j^p - u_j^r)| + |\mu_j (u_j^r - u_j^q)| \\ &= \|\boldsymbol{\mu} \odot (\boldsymbol{u}^p - \boldsymbol{u}^r)\|_1 + \|\boldsymbol{\mu} \odot (\boldsymbol{u}^r - \boldsymbol{u}^q)\|_1. \end{aligned}$$

Working similarly, we can show that

$$\begin{aligned} \|\boldsymbol{\mu} \odot (\boldsymbol{l}^p - \boldsymbol{l}^q)\|_1 &\leq \|\boldsymbol{\mu} \odot (\boldsymbol{l}^p - \boldsymbol{l}^r)\|_1 + \|\boldsymbol{\mu} \odot (\boldsymbol{l}^r - \boldsymbol{l}^q)\|_1, \\ \|\boldsymbol{\nu} \odot (\boldsymbol{C}^p - \boldsymbol{C}^q)\|_1 &\leq \|\boldsymbol{\nu} \odot (\boldsymbol{C}^p - \boldsymbol{C}^r)\|_1 + \|\boldsymbol{\nu} \odot (\boldsymbol{C}^r - \boldsymbol{C}^q)\|_1. \end{aligned}$$

Moreover, we can show by case analysis that

$$\lambda \cdot \mathbb{1}_{(k^p \neq k^q)} \leq \lambda \cdot \mathbb{1}_{(k^p = k^r)} + \lambda \cdot \mathbb{1}_{(k^r = k^q)}.$$

By summing the above four inequalities, we get that $d(p,q)$ satisfies the triangle inequality (axiom **(iv)**) $d(p,q) \leq d(p,r) + d(r,q)$, and is therefore a metric. □

We now proceed with the proof of Lemma 8: Proof of Lemma 8 Let $\mathcal{T}$ denote the set of all trees of maximum depth D and $\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3 \in \mathcal{T}$ with $\mathbb{T}_1 \neq \mathbb{T}_2 \neq \mathbb{T}_3$. We assume, without loss of generality, that $\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3$ have the same number of paths P;

for the more general case, we work similarly to the proof of Corollary 1 (by appending "dummy" paths to the trees with the fewer paths).

We need to show that **(i)** $d(\mathbb{T}_1, \mathbb{T}_1) = 0$, **(ii)** $d(\mathbb{T}_1, \mathbb{T}_2) > 0$, **(iii)** $d(\mathbb{T}_1, \mathbb{T}_2) = d(\mathbb{T}_2, \mathbb{T}_1)$, **(iv)** $d(\mathbb{T}_1, \mathbb{T}_3) \leq d(\mathbb{T}_1, \mathbb{T}_2) + d(\mathbb{T}_2, \mathbb{T}_3)$. Axiom **(i)** is satisfied by matching each path in Problem (3.3) with itself and then applying Lemma 10. Axiom **(ii)** is satisfied by observing that $\mathbb{T}_1 \neq \mathbb{T}_2$ (implying they differ in at least one path) and then applying Lemma 10. Axiom **(iii)** is satisfied by again invoking Lemma 10 and noticing that the solution to Problem (3.3) does not depend on the order of the input trees.

We now prove axiom **(iv)**. Let us index the paths in $\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3$ according to the optimal matching between $\mathbb{T}_1$ and $\mathbb{T}_2$ so that $p_1^1 \leftrightarrow p_1^2, \ \ldots, \ p_P^1 \leftrightarrow p_P^2$, and between $\mathbb{T}_2$ and $\mathbb{T}_3$ so that $p_1^2 \leftrightarrow p_1^3, \ \ldots, \ p_P^2 \leftrightarrow p_P^3$; the subscript corresponds to the path index and the superscript corresponds to the tree index. Let us also consider a permutation of the paths in $\mathbb{T}_3$ (denoted by q) according to the optimal matching between $\mathbb{T}_1$ and $\mathbb{T}_3$ so that $p_1^1 \leftrightarrow q_1^3, \ \ldots, \ p_P^1 \leftrightarrow q_P^3$. By application of Lemma 10 we get that, for any $i \in [P]$,

$$d(p_i^1, p_i^2) + d(p_i^2, p_i^3) \geq d(p_i^1, p_i^3) \tag{3.5}$$

Taking the sum across all paths, yields

$$\begin{aligned} d(\mathbb{T}_1, \mathbb{T}_2) + d(\mathbb{T}_2, \mathbb{T}_3) &\overset{\text{(a)}}{=} \sum_{i \in [P]} d(p_i^1, p_i^2) + d(p_i^2, p_i^3) \overset{\text{(b)}}{\geq} \sum_{i \in [P]} d(p_i^1, p_i^3) \\ &\overset{\text{(c)}}{\geq} \sum_{i \in [P]} d(p_i^1, q_i^3) \overset{\text{(d)}}{=} d(\mathbb{T}_1, \mathbb{T}_3), \end{aligned}$$

where **(a)** follows from the fact that $p_i^1 \leftrightarrow p_i^2$ is the optimal matching between $\mathbb{T}_1$ and $\mathbb{T}_2$, and $p_i^2 \leftrightarrow p_i^3$ is the optimal matching between $\mathbb{T}_2$ and $\mathbb{T}_3$, so summing the matched path distances gives the optimal objective value of Problem (3.3), that is, the tree distance; **(b)** follows from Equation (3.5); **(c)** follows from the fact that $p_i^1 \leftrightarrow q_i^3$ is the optimal matching between $\mathbb{T}_1$ and $\mathbb{T}_3$ whereas $p_i^1 \leftrightarrow p_i^3$ is not; **(d)** follows from the definition of the tree distance (Problem (3.3)). Therefore, $d(\mathbb{T}_1, \mathbb{T}_2)$ satisfies the triangle inequality (axiom **(iv)**) and is a metric. □

3.6.2 Proof of Corollary 1

Proof of Corollary 1. Given trees $\mathbb{T}_1$ and $\mathbb{T}_2$ with $T_1 > T_2$, we append $T_1 - T_2$ "dummy paths" to $\mathbb{T}_2$, which we denote by $\mathcal{D}(\mathbb{T}_2)$, such that the distance from any dummy path to any path $p \in \mathcal{P}(\mathbb{T}_1)$ is equal to $w(p)$. That is, $d(p, q) = w(p),\ \forall p \in \mathcal{P}(\mathbb{T}_1), q \in \mathcal{D}(\mathbb{T}_2)$. We refer to this representation for Problem (3.3) as Problem (3.6). We then rewrite the linear relaxation of Problem (3.6) as:

$$
\begin{aligned}
d(\mathbb{T}_1, \mathbb{T}_2) \;=\; \min_{\boldsymbol{x}} \quad & \sum_{p \in \mathcal{P}(\mathbb{T}_1)} \sum_{q \in \mathcal{P}(\mathbb{T}_2) \cup \mathcal{D}(\mathbb{T}_2)} d(p, q) x_{pq} \\
\text{s.t.} \quad & \sum_{q \in \mathcal{P}(\mathbb{T}_2)} x_{pq} = 1, \quad \forall p \in \mathcal{P}(\mathbb{T}_1) \\
& \sum_{p \in \mathcal{P}(\mathbb{T}_1)} x_{pq} = 1, \quad \forall q \in \mathcal{P}(\mathbb{T}_2) \\
& x_{pq} \geq 0, \qquad \forall p \in \mathcal{P}(\mathbb{T}_1), \quad \forall q \in \mathcal{P}(\mathbb{T}_2) \cup \mathcal{D}(\mathbb{T}_2)
\end{aligned}
\tag{3.7}
$$

The coefficient matrix of Problem (3.7) is totally unimodular and thus every extreme point of the feasible region is integral. Moreover, since Problem (3.7) is a linear optimization problem, the optimum will be attained at an extreme point. Therefore, there exists an optimum to Problem (3.7) that is integral. Since Problem (3.7) is more constrained than Problem (3.6), the integral optimum to the former also solves the latter, as well as the original Problem (3.3). The cases with $T_1 \leq T_2$ can be proved similarly. □

3.6.3 Proof of Lemma 9

Proof of Lemma 9. The maximum number of paths in a tree of depth D is 2^D. The maximum distance between two paths of depth D is $2D + \lambda$: it occurs when the D splits are performed on $2D$ distinct features, each split is performed on values that are ϵ-close to the upper or lower bound for the corresponding feature (with $\epsilon \to 0$), and the resulting class labels are also different. Provided that the number of features and classes are large enough, we can construct two trees that achieve this upper bound. □

3.7 Extended Sensitivity Analysis of the Proposed Tree Distance

We present the full numerical study of the key properties of the proposed tree distance metric, which we briefly described in Section 3.2.6. We use the case studies of Section 3.4. We test the sensitivity of the proposed distance metric to two types of perturbations — direct and indirect ones. Direct perturbations refer to immediate interventions and changes in the tree structure. Indirect perturbations refer to modifications in the training data. In the remainder of this section, we more concretely describe the methodology we use to obtain each type of perturbation and discuss the corresponding results, shown in Figure 3-8.

To investigate the sensitivity of the proposed distance metric to direct perturbations, we repeat the following process 100 times independently for each case study. In each repetition, we train a decision tree model using 5-fold cross-validation. We then randomly perturb each threshold t in the tree by a percentage θ drawn uniformly at random from $[0, \theta_{\max}]$, i.e., $t_{\text{pert}} = t \cdot (1 + 2 \cdot \theta_{\max} \cdot \theta - \theta_{\max})$. We vary $\theta_{\max} \in [0, 1]$ and measure the distance between the original and the perturbed trees in each case. We report the mean and standard deviation of the distance, obtained across all repetitions and case studies. Figure 3-8(left) suggests that, as the amount of perturbation increases, the distance increases monotonically. The relationship is linear and the mean distance, expressed as a percentage of the maximum possible distance between any two trees of the given depth, varies between 1.5%, when the maximum perturbation is 10%, and 12.5%, when the maximum perturbation is 100%.

We proceed to study the sensitivity of the proposed distance metric to indirect perturbations, which refer to modifications in the training data. We repeat the following process 10 times for each case study. In each repetition, we randomly permute the data and keep the first half of them to train a decision tree model using 5-fold cross-validation. Then, for $\theta \in [0.2, \ldots, 1.]$, we sequentially replace a θ fraction of the first half of the data with the same amount of data taken from the second half, we train a new decision tree model, and we measure the distance between the

two. We report the mean and standard deviation of the distance, obtained across all repetitions and case studies. Figure 3-8(right) shows that, as θ increases, the distance has an increasing trend, albeit non-monotonic. We attribute this lack of monotonicity to the inherent instability of decision tree models [65], rather than to the proposed distance metric. We remark that the distance percentages in this experiment are low because the upper bound on the distance was obtained using the largest maximum allowable tree depth examined during cross-validation (rather than the actual depth of any of the two trees).

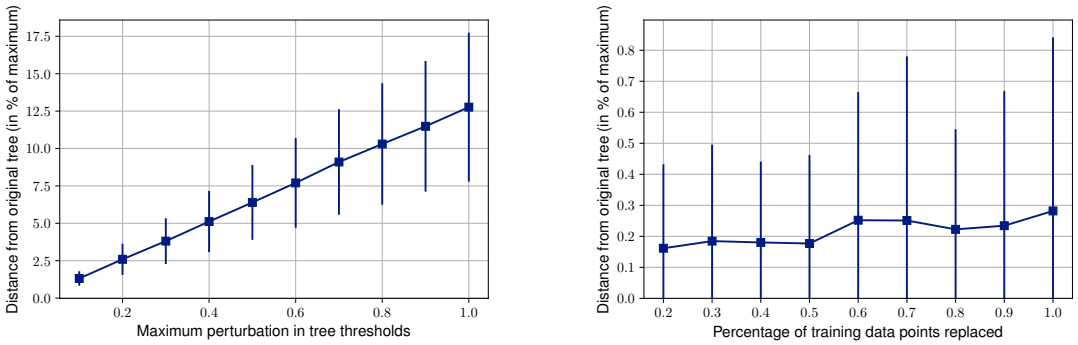


Figure 3-8: Mean and standard deviation of the distance between the original tree and perturbed tree. The perturbed trees are obtained by randomly perturbing the split thresholds (left) or replacing a fraction of the training points (right). We aggregate across multiple repetitions (i.e., perturbations) for each dataset.

Chapter 4

Volume and the Backbone Method

4.1 Introduction

In the Big Data era, the scalability of machine learning models is constantly challenged. *Ultra-high dimensional* datasets are present in a variety of applications, ranging from physical sciences and engineering to social sciences and medicine. From a practical standpoint, we consider a dataset to be ultra-high dimensional when the number of features p exceeds the number of data points n and is at least two orders of magnitude greater than the known scalability limit of the learning task at hand; as we discuss in the sequel, this limit is $p \sim 10^5$ for the sparse regression algorithm that we use, and $p \sim 10^3$ for the decision tree induction algorithm. From a theoretical standpoint, we assume that $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$.

A commonly held assumption to address high dimensional regimes is that the underlying model is *sparse*, that is, among all input features, only $k < n \ll p$ (e.g., $k \leq 100$) are relevant for the task at hand [128]. For instance, in the context of regression, the sparsity assumption implies that only a few regressors are set to nonzero level. In the context of decision trees, it means that most features do not appear in any split node. The sparsity requirement can either be explicitly imposed via a cardinality constraint or penalty, as in sparse regression (also known as best subset selection), or can implicitly be imposed by the structure of the model, as in depth-constrained decision trees, where the number of features that we split on

is upper bounded as function of the tree depth. In general, sparsity is a desirable property for machine learning models, as it makes them more interpretable and often improves their generalization ability [174].

Learning a *sparse machine learning model* can naturally be formulated as a *mixed integer optimization* (MIO) problem by associating each feature with an auxiliary binary decision variable that is set to 1 if and only if the feature is identified as relevant [34]. Despite the -theoretical- computational hardness of solving MIO problems, the remarkable progress in the field has motivated their use in a variety of machine learning problems, ranging from sparse regression and decision trees to principal component analysis and matrix completion [31]. The rich modeling framework of MIO enables us to directly model the problem at hand and often compute provably optimal or near-optimal solutions using either exact methods (e.g., branch and bound) or tailored heuristics (e.g., local search). Moreover, the computational efficiency and scalability of MIO-based methods is far beyond what one would imagine a decade ago. For example, we are able to solve sparse regression problems with $p \sim 10^5$ features [35] and induce decision trees for problems with $p \sim 10^3$ features [45]. Nevertheless, MIO-based formulations are still *challenged in ultra-high dimensional regimes* as the ones we examine.

The *standard way of addressing ultra-high dimensional problems* is to perform either a screening (i.e., filtering) step [100], that eliminates a large portion of the features, or more sophisticated dimensionality reduction methods: **(a)** feature selection methods, that aim to select a subset of relevant features, or **(b)** feature extraction methods, that project the original features to a new, lower-dimensional feature space; see [125, 154] for detailed reviews. After the dimensionality reduction step, we can solve the MIO formulation for the reduced problem, provided that sufficiently many features have been discarded. However, commonly used screening (or, more generally, feature selection) approaches are heuristic and often lead to suboptimal solutions, whereas feature extraction approaches are uninterpretable in terms of the original problem's features.

In this chapter, we introduce the *backbone method for sparse supervised learning*, a

two-phase framework that, as we empirically show, scales to ultra-high dimensions (as defined earlier) and provides significantly higher quality solutions than screening approaches, without sacrificing interpretability (as opposed to feature extraction methods). In the first phase of our proposed method, we aim to identify the "backbone" of the problem, which consists of features that are likely to be part of an optimal solution. We do so by collecting the solutions to a series of subproblems that are more tractable than the original problem and whose solutions are likely to contain relevant features. These subproblems also need to be "selective," in the sense that only the fittest features survive after solving each of them. In the second phase, we solve the MIO formulation considering only the features that have been included in the backbone. Although our framework is generic and can be applied to a large variety of problems, we focus on two particular problems, namely sparse regression and decision trees, and illustrate how the backbone method can be applied to them. We accurately solve sparse regression problems with 10^7 features in minutes and 10^8 features in hours, as well as decision tree problems with 10^5 features in minutes, that is, two orders of magnitude larger than the known scalability limit of methods that solve to optimality or near-optimality the actual MIO formulation for each problem.

Our proposed backbone method was inspired by a large-scale vehicle routing application; using a backbone algorithm, [33] solve, in seconds, problems with thousands of taxis serving tens of thousands of customers per hour. In this work, we develop the backbone method for sparse supervised learning in full generality. Moreover, we remark that the method can be applied to a variety of machine learning problems that exhibit sparse structure, even beyond supervised learning. For example, in the clustering problem, a big portion of pairs of data points will never be clustered together in any near optimal cluster assignment and hence we need not consider assigning them to the same cluster. Finally, we note that, within the sparse supervised learning framework that we examine in this chapter, the backbone method can also be used as a feature selection technique in combination with *any sparsity-imposing but not necessarily MIO-based method*.

4.1.1 Relevant Literature: Sparse Regression in Ultra-High Dimensions

In this section, we briefly review the landscape of the sparse regression problem in ultra-high dimensions. As a reminder, a regression problem is considered *high dimensional* when the rank of the design matrix $\boldsymbol{X}$ is smaller than the number of features p, i.e., $\text{rank}(\boldsymbol{X}) < p$; this is, for example, the case when n, the number of data points in the dataset, satisfies $n < p$. In such regimes, sparsity provides a way around this limitation and, in addition, hopefully leads to more interpretable models, where only k features actually affect the prediction. In many cases, k is determined by the application, [143] address such concerns by proposing efficient cross validation strategies. In this chapter, we assume that k is known and given; the combination of efficient cross validation procedures with our proposed method is straightforward.

Screening. Sure independence screening (SIS) is a two-phase learning framework, introduced by [100, 101]. In the first phase, SIS ranks the features based on their marginal utilities; e.g., for linear regression, the proposed marginal utility is the correlation between each feature and the response. By keeping only the highest-ranked features, SIS guarantees that, under certain conditions, all relevant features are selected with high probability. The screening procedure can be implemented iteratively, conditioned on the estimated set of features from the previous step. In the second phase, SIS conducts learning and inference in the reduced feature space consisting only of the selected features, using surrogate formulations such as lasso. Among the various other extensions of SIS, we emphasize the works of [102] and [103] that extend SIS to generalized linear models, as well as [104] and [175] that extend SIS beyond the linear model. [18] propose safe screening rules for the exact sparse regression formulation, derived from a convex relaxation solution; such rules eliminate features based on guarantees that a feature may or may not be selected in an optimal solution. In this chapter, we aim to address substantially bigger problems, so our proposed framework does not require solving a convex relaxation of the entire problem.

Distributed Approaches. An alternative path to address large scale sparse regression problems is using distributed methods. The ADMM framework [61] can be used to fit, in a distributed fashion, a regression model on vertically partitioned data, provided that the regularizer is separable at the level of the blocks of features; this is, e.g., the case in lasso. The DECO framework of [219], after arbitrarily partitioning the features, performs a decorrelation step via the singular value decomposition of the design matrix, fits a lasso model in each feature subset, and combines the estimated coefficients are centrally. Several hybrid methods that combine screening with ideas from the distributed literature have also been developed [231, 238, 202].

4.1.2 Relevant Literature: Decision Trees in Ultra-High Dimensions

Decision trees are known to suffer from the curse of dimensionality and to not perform well in the presence of many irrelevant features [13]. Among the most notable attempts to scale decision trees to ultra-high dimensional problems is the recent work by [158], who develop a sparse version of the perceptron decision trees framework [29] and solve problems with $p \sim 10^6$ features.

4.1.3 Relevant Literature: Backbones in Optimization

[195], [215], and the many references therein use the term backbone of a discrete optimization problem to refer to a set of frozen decision variables that *must be set to one in any optimal or near-optimal solution*. Thus, the backbone can be obtained as the intersection of all optimal solutions to the problem. For example, in the satisfiability decision problem, the backbone of a formula is the set of literals which are true in every model. This definition is fundamentally different from our notion of a backbone and, from a practical perspective, identifying such backbone can be as hard as solving the actual problem. In our approach, the term backbone refers to all variables that *are set to one in at least one near-optimal solution*. Thus, the backbone can be obtained as the union of all near-optimal solutions. To the best of

our knowledge, the notion of backbone that we examine in this chapter first appeared in [33] in the context of online vehicle routing.

4.1.4 Contributions and Outline

Our key contributions can be summarized as follows:

- We develop the backbone method, a novel, two-phase framework that, as we empirically show, performs highly effective feature selection and enables sparse machine learning models to scale to ultra-high dimensions. As mixed integer optimization offers a natural way to model sparsity, we focus on MIO-based machine learning methods. Nonetheless, our framework is developed in full generality and can be applied to any sparsity-inducing machine learning method.

- We apply the backbone method to the sparse regression problem. We show that, under certain assumptions and with high probability, the backbone set consists of the true relevant features. Our computational results on both synthetic and real-world data indicate that the backbone method outperforms or competes with state-of-the-art methods for ultra-high dimensional problems, accurately scales to problems with $p \sim 10^7$ in minutes and problems with $p \sim 10^8$ in hours. In problems with $p \sim 10^5$, where exact methods apply and hence we can compare with near-optimal solutions, the backbone method, by drastically reducing the problem size, achieves, faster (by $15 - 20\%$) and with fewer data points, the same levels of accuracy as exact methods, while providing optimality guarantees, albeit for the reduced problem.

- We apply the backbone method to the decision tree problem. Our computational results indicate that the backbone method scales to problems with $p \sim 10^5$ in minutes, outperforms CART, and competes with random forest, while still outputting a single, interpretable tree. In problems with $p \sim 10^3$, the backbone method can accurately filter the feature set and compute, substantially faster (by $10 - 100$ times), decision trees with comparable out-of-sample performance

to that of the decision tree obtained by applying the optimal trees framework to the entire problem.

The structure of the chapter is as follows. In Section 4.2, we develop the generic framework for the backbone method in a general supervised learning setting; we unfold the components of the method and discuss topics that include hyperparameter selection, termination, complexity, and parallel implementation. In Section 4.3, we apply the backbone method to the sparse regression problem, discuss several implementation aspects and challenges, and present a theoretical result on the method's accuracy, which can provide guidance in tuning the method's hyperparameters. In Section 4.4, we apply the backbone method to the decision tree problem and again discuss several implementation aspects and challenges. Section 4.5 investigates, via experiments on synthetic datasets, the backbone method's scalability, performance in various regimes, and sensitivity to the method's hyperparameters and components. In Section 4.6, we evaluate the method through computational experiments on real-world datasets. Finally, Section 4.7 concludes the chapter.

4.2 The Backbone Method

In this section, we describe the backbone method in a general supervised learning context. The main idea is that many real-world ultra-high dimensional machine learning problems are indeed sparse, which means that most features are uninformative for performing regression or classification and hence need not be considered when solving the actual problem's formulation. Therefore, the core of the two-phase backbone method is the construction of the *backbone set*, which consists of features identified as potentially relevant. The two phases of the backbone method are:

1. **Backbone Construction Phase:** Form a series of tractable subproblems that can be solved efficiently while still admitting sparse solutions, by reducing the dimension and/or relaxing the original problem. Construct the backbone set by collecting all decision variables that participate in the solution to at least one subproblem.

2. **Reduced Problem Solution Phase:** Solve the reduced problem to near-optimality, considering only the decision variables that were included in the backbone set.

The backbone method differs from existing feature selection methods in that we propose to filter uninformative features via a procedure closely related to the actual problem. For example, to perform feature selection for the decision tree problem, we train more tractable decision trees in the subproblems and select those features that are identified as relevant within the subproblems' trees. A major difference between the backbone method and heuristic solution methods is that we aim to reduce the problem dimension as much as possible so that the resulting reduced problem can actually be solved to near-optimality. From an optimization perspective, the backbone method can loosely be viewed as a heuristic branch and price approach.

4.2.1 A Generic Hierarchical Backbone Algorithm

We proceed by applying the backbone method in a general sparse supervised learning setting (Algorithm 3). Let $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ be the input data (n and p denote the number of data points and features respectively) and $\boldsymbol{y}$ the vector of responses (for regression problems) or class labels (for classification problems). At the end of the backbone construction phase, the backbone set, which we denote by $\mathcal{B}$, must be small enough so that we can actually solve the resulting reduced problem. Therefore, we implement the backbone construction phase in Algorithm 3 hierarchically, in a bottom-up divide and conquer fashion, until the size of the backbone set is sufficiently small.

The notation used in Algorithm 3 also includes the following. The set $\mathcal{U}_t$ consists of all features that are candidates to enter the backbone during iteration t. Initially, all features are considered, so $\mathcal{U}_0 = [p] = \{1, ..., p\}$ (unless the `screen` function, which will be explained shortly, is used). When the data matrix $\boldsymbol{X}$ is indexed by a set $\mathcal{C} \subseteq [p]$, denoted by $\boldsymbol{X}_{\mathcal{C}}$, it is implied that only the features contained in $\mathcal{C}$ have been selected.

As part of the input to Algorithm 3, we need to specify the application-specific

functions that will be used within our framework. In particular, these include:

- Function `screen`: The function used to eliminate features that are highly likely to be irrelevant. This function is optionally called before the backbone construction phase. The output of this function is the set of features that have not been eliminated, along with their marginal utilities.

- Function `construct_subproblems`: The function used to construct more tractable subproblems. Since p is assumed to be very large, the `construct_subproblems` function acts by selecting a subset of features $\mathcal{P}$ for each subproblem. Thus, the output of this function is a collection of features for every subproblem.

- Function `fit_subproblem`: The model fitting function to be used within each subproblem. Fits a sparse model of choice (regression, trees, etc.) to the data that are given as input to the subproblem. This function has to be highly efficient for the backbone construction to be fast, so we propose that a relaxed/surrogate formulation or heuristic solution method is used. For simplicity in notation, we assume that all input hyperparameters (e.g., for lasso, the regularization weight/sequence of weights) are "hidden" within the definition of the function. The output of this function is the learned model.

- Function `extract_relevant`: The function that takes a sparse model as input and extracts all features that the model identifies as relevant.

- Function `fit`: The target fitting function that fits a sparse model of choice to the data given as input. As this function is to be applied to the (small) backbone set, it needs not be extremely efficient, so we propose that a method that comes with optimality guarantees is used. For simplicity in notation, we assume that all input hyperparameters are "hidden" within the definition of the function. The output of this function is the learned model.

Algorithm 3 has a number of hyperparameters; in the sequel, we discuss how the hyperparameters can be tuned in practice (Section 4.2.4) and in theory (Section 4.3.2),

and examine how sensitive the backbone method is with respect to each of them (Section 4.5.7). The hyperparameters are the following:

- $M \in \mathbb{N}$: The number of subproblems to solve in each iteration (or hierarchy).

- $\beta \in (0, 1]$: The fraction of features to include in each subproblem's feature set.

- $\alpha \in (0, 1]$: The fraction of features to keep after applying the `screen` function.

- $B_{\max} \in \mathbb{N}$: The maximum allowable backbone size.

Algorithm 3: Generic Hierarchical Backbone Algorithm

Input: Data $(\boldsymbol{X}, \boldsymbol{y})$, hyperparameters $(M, \beta, \alpha, B_{\max})$, functions (`screen`, `construct_subproblems`, `fit_subproblem`, `extract_relevant`, `fit`).
Output: Learned model.

$t \leftarrow 0$
$\mathcal{U}_0, \boldsymbol{s} \leftarrow \texttt{screen}(\boldsymbol{X}, \boldsymbol{y}, \alpha)$
repeat
  $\mathcal{B} \leftarrow \emptyset$
  $(\mathcal{P}_m)_{m \in [M]} \leftarrow \texttt{construct\_subproblems}(\mathcal{U}_t, \boldsymbol{s}, \lceil \frac{M}{2^t} \rceil, \beta)$
  for $m \in [M]$ **do**
    $\text{model}_m \leftarrow \texttt{fit\_subproblem}(\boldsymbol{X}_{\mathcal{P}_m}, \boldsymbol{y})$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \texttt{extract\_relevant}(\text{model}_m)$
  end for
  $t \leftarrow t + 1$
  $\mathcal{U}_t \leftarrow \mathcal{B}$
until $|\mathcal{B}| \leq B_{\max}$ (or other termination criterion is met)
model $\leftarrow \texttt{fit}(\boldsymbol{X}_{\mathcal{B}}, \boldsymbol{y})$
return model

The backbone method provides a generic framework that can be used to address a large number of different problems. As illustrated by the applications which we investigate in the following sections, depending on the problem at hand, there are several design choices that affect the performance of the method. For example:

- How to form the subproblems, so that they are tractable and, at the same time, useful for the original problem? We address this question in Section 4.2.3 by developing a generic framework for regression and classification problems.

- How to solve the subproblems and, in particular, what is the impact of applying heuristics within each subproblem for the quality of the backbone set? We examine these questions for the sparse regression and the decision tree problems in Sections 4.3 and 4.4, respectively.

4.2.2 The `screen` Function: Sure Independence Screening at a Glance

In this section, we describe the `screen` component of the backbone method. We first briefly present the sure independence screening (SIS) framework [100], which plays an important role in our approach.

SIS is a two-phase sparse learning framework. In the first phase, we rank the features based on their marginal utilities s_j and retain only the top d features, i.e.,

$$\mathcal{M} = \{1 \leq j \leq p : \ s_j \text{ is among the top } d \text{ largest ones}\}.$$

Given a convex loss function ℓ (e.g., ordinary least squares or logistic loss), we measure the marginal utility s_j of each feature $j \in [p]$ using the empirical maximum marginal likelihood estimator, i.e.,

$$(\hat{w}_{0,j}, \hat{w}_j) = \text{argmin}_{w_j^0, w_j} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, w_j^0 + w_j X_{i,j}). \tag{4.1}$$

We then estimate the marginal utility of feature j as either the magnitude of the maximum marginal likelihood estimator $s_j := |\hat{w}_j|$ or the minimum of the loss function $s_j := \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{w}_{0,j} + \hat{w}_j X_{i,j})$. In practice, we usually choose d using cross validation. In the second phase, we conduct learning and inference in the reduced feature space consisting only of features in $\mathcal{M}$. Under certain conditions, SIS possesses the sure screening property, which requires that all relevant features are contained in the set $\mathcal{M}$ with probability tending to one [100, 102, 103].

We present the `screen` function that we use within our framework in Algorithm 4. Our proposed `screen` function computes the marginal utility of each feature and

Algorithm 4: Function `screen`

Input: Data $(\boldsymbol{X}, \boldsymbol{y})$, hyperparameter α.
Output: Screened features $\mathcal{M}$, marginal utilities $\boldsymbol{s}$.

$\boldsymbol{s} \leftarrow \mathbf{0}_p$
for $j \in [p]$ **do**
  $(\hat{w}_{0,j}, \hat{w}_j) \leftarrow \text{argmin}_{w_j^0, w_j} \frac{1}{n} \sum_{i=1}^n \ell(y_i, w_j^0 + w_j X_{i,j})$
  $s_j \leftarrow |\hat{w}_j|$
end for
$\boldsymbol{r} \leftarrow \texttt{arg\_sort}(\boldsymbol{s})$ ▷ Return permutation of $[p]$ that sorts $\boldsymbol{s}$ in decreasing order.
$\mathcal{M} \leftarrow \{r_i\}_{i=1}^{\lceil \alpha p \rceil}$ ▷ Keep top $\lceil \alpha p \rceil$ features.
return $\mathcal{M}, \boldsymbol{s}$

eliminates the $(1 - \alpha)p$ lowest ranked features. Concerning the choice of the loss function ℓ in Equation (4.1) or, more generally, the scoring function we use, we make the following remarks:

- In linear regression problems, we use the squared loss function [100]. By doing so, the marginal utility measure given by $s_j := |\hat{w}_j|$ simplifies, under standard assumptions such as standardizing the data, to the absolute marginal empirical correlation between feature j and the response, namely, $s_j := |\widehat{\text{cor}}(\boldsymbol{X}_j, \boldsymbol{y})|$ (note that $\widehat{\text{cor}}$ represents the empirical correlation).

- In binary classification problems, we use the logistic loss function, whereas in multi-class classification problems, the multi-category SVM loss function [102].

- In highly nonlinear problems, we use the nonparametric screening approach by [104] or the entropy-based approach by [175], which is particularly suited for decision tree problems.

This approach provides a unified scoring framework for both regression and classification problems. Since we only use `screen` as a preprocessing step, we do not perform cross validation and, instead, select a slightly larger value for α; e.g., we pick α such that $\alpha p \sim 10n$, which is consistent with the theoretical and empirical analysis of [100] and, as we empirically show, is a good choice in practice. As we discussed in the Introduction of this chapter, by using feature marginal utility measures that

satisfy the sure screening property, we obtain strong theoretical guarantees that we will not eliminate any relevant feature during this step.

4.2.3 The `construct_subproblems` Function

In this section, we describe the `construct_subproblems` component of the backbone method, which aims to construct more tractable subproblems.

Having computed each feature's marginal utility as per Section 4.2.2, we construct the feature set of subproblem $m \in M$ by sampling $\lceil \beta\alpha p \rceil$ features among the $\lceil \alpha p \rceil$ features that survived the screening step of Algorithm 4. Within each subproblem, we sample the features without replacement and with probability that increases exponentially according to each feature's marginal utility. Algorithm 5 provides pseudocode for the proposed approach.

Algorithm 5: Function `construct_subproblems`

Input: Candidate features $\mathcal{U}$, feature marginal utilities $\boldsymbol{s}$, hyperparameters M, β.
Output: Subproblem feature sets $(\mathcal{P}_m)_{m\in[M]}$.

$\boldsymbol{\pi} \leftarrow \mathbf{0}_{|\mathcal{U}|}$
for $j \in \mathcal{U}$ **do**
  $s_j \leftarrow \frac{s_j}{\max_{i\in\mathcal{U}} s_i}$ ▷ Normalize utilities.
  $\pi_j \leftarrow \exp(s_j + 1)$
end for
$(\mathcal{P}_m)_{m\in[M]} \leftarrow (\emptyset)_{m\in[M]}$
for $m \in [M]$ **do**
  $\mathcal{P}_m \leftarrow \texttt{sample}(\mathcal{U}, \beta|\mathcal{U}|, \boldsymbol{\pi})$ ▷ Sample $\beta|\mathcal{U}|$ features from $\mathcal{U}$, each w/ prob $\propto \pi_j$.
end for
return $(\mathcal{P}_m)_{m\in[M]}$

The main benefits of breaking the problem into subproblems, as per Algorithm 5, after -possibly- applying the screening step of Algorithm 4, are as follows:

- *Tractability*: Since p is assumed to be very large and $\lceil \alpha p \rceil$ can also be large, by selecting a subset of features $\mathcal{P}$ for each subproblem, we perform an additional dimensionality reduction step. In particular, assume that all k relevant features survived the screening step and that we randomly sample each subproblem's $\lceil \beta\alpha p \rceil$ features. Then, in expectation, we will have βk relevant features within

each subproblem. Thus, the subproblems clearly become more tractable, since, even with an exhaustive search procedure, we end up having to check $\binom{\beta\alpha p}{\beta k}$ subsets of features, instead of the initial $\binom{\alpha p}{k}$ or $\binom{p}{k}$, which can be astronomically larger.

- *Ensemble learning*: Our proposed approach, of sampling the features that form each subproblem's feature set with probability that increases exponentially according to each feature's marginal utility, is partly inspired by the exponential mechanism of [163]. We aim to bridge the gap between the popular random subspace method [135] and its variants, whereby, within each subproblem, a feature subset is selected uniformly at random, and the deterministic subspace method by [148], which ranks features according to a predefined function and selects the top ones.

  Using our proposed approach, we still enjoy the benefits of ensemble learning, whereby each feature gets examined multiple times and as part of different feature sets, while, at the same time, we avoid having subproblems with few or no relevant features. The absence of relevant features from a subproblem's feature set makes the subproblem harder to solve, since, any relevant feature that is not sampled and hence not contained within the subproblem's feature set, is essentially viewed as noise within this subproblem. In Section 4.5.7, we empirically show that our proposed approach has an edge over sampling features uniformly at random or with probabilities proportional to their screening scores.

4.2.4 Discussion: Hyperparameters, Termination, Complexity, and Parallel Implementation

In this section, we discuss several other aspects of the backbone method: how we choose the hyperparameter values in practice, how we ensure the algorithm's termination, what is the computational complexity of the method, and how the algorithm can be implemented in parallel.

Hyperparameter selection. The hyperparameters $M, \beta, \alpha, B_{\max}$ of the backbone method can in practice be tuned via cross validation using a multi dimensional grid search. However, we remark that, in practical applications, β and $B_{\max}$ are partly determined by the available computational resources (e.g., available memory) and, more specifically, by the size of the problems that we can solve using the `fit_subproblem` function and the `fit` function, respectively. The hyperparameter α, which determines how many features the `screen` function will eliminate, is also partly dependent on the available resources, although we do have more freedom in tuning it; [100] provide both theoretical and empirical insights on how to choose α. Finally, the number of subproblems M can be selected dynamically: we pick a large value for M; we keep solving subproblems and add their solutions to the backbone set; if the backbone set does not change after two consecutive subproblems, we stop.

In Section 4.3.2, we provide theoretical guidance on how to choose M and α as function of the remaining problem and algorithm parameters. In Section 4.5.7, we present a detailed sensitivity analysis of the backbone method with respect to each of the hyperparameters.

Termination. To ensure finite termination of Algorithm 3, we require that the `fit_subproblem` function returns a model with at most $k_{\max} \leq k$ relevant features. In the context of sparse regression, we can directly control $k_{\max}$. In the context of decision trees, we indirectly control $k_{\max}$ via the tree's depth. Further, we reduce the number of subproblems to be solved in each iteration by a factor of 2. Therefore, the number of iterations that Algorithm 3 will perform is at most

$$H = \left\lceil \log_2 \frac{M k_{\max}}{B_{\max}} + 1 \right\rceil.$$

Computational Complexity. We denote by $f_{\texttt{screen}}(n, p)$, $f_{\texttt{construct}}(p, \beta)$, and $f_{\texttt{solve}}(n, p)$ the complexity of the `screen` function, the `construct_subproblems` function, and the `solve_subproblem` function, respectively. Then, the overall complexity

of the backbone construction phase of Algorithm 3 is

$$\mathcal{O}\left[f_{\texttt{screen}}(n,p) + \sum_{t=1}^{H} \frac{M}{2^{t-1}} f_{\texttt{construct}}(\alpha p, \beta) + \sum_{t=1}^{H} \frac{M}{2^{t-1}} f_{\texttt{solve}}(n, \alpha\beta p)\right]$$
$$= \mathcal{O}\left[f_{\texttt{screen}}(n,p) + M f_{\texttt{construct}}(\alpha p, \beta) + M f_{\texttt{solve}}(n, \alpha\beta p)\right].$$

As an example, consider a sparse regression problem where we use the empirical correlation between each feature and the response as the scoring function for `screen`, use a simple heap-sampling implementation for `construct_subproblems`, and solve subproblems using the LARS algorithm for the lasso formulation [95]. Then, the resulting complexity is $\mathcal{O}\left[np + M\alpha\beta p \log(\alpha p) + M(\alpha\beta p)^2 n\right]$.

Parallel & Distributed Implementation. An important feature of the backbone method is that it can be naturally executed in parallel by assigning different subproblems to different computing nodes and then centrally collecting all solutions and solving the reduced problem. This is particularly important in distributed applications, where a massive dataset is "vertically partitioned" among different nodes, i.e., each node has access to a subset of the features in the data. This regime, despite being common in practice, has not been considered much in the literature.

4.3 The Backbone of Sparse Regression

Given data $\{(\boldsymbol{x}_i, y_i)\}_{i\in[n]}$, where, $\forall i \in [n]$, $\boldsymbol{x}_i \in \mathbb{R}^p$ (with $n \ll p$), and $y_i \in \mathbb{R}$ (for regression) or $y_i \in \{-1, 1\}$ (for binary classification), and a convex loss function ℓ, consider the sparse empirical risk minimization problem with Tikhonov regularization and an explicit sparsity constraint, outlined as

$$\min_{\boldsymbol{w}\in\mathbb{R}^p} \quad \sum_{i=1}^{n} \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i) + \frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2 \qquad \text{s.t.} \quad \|\boldsymbol{w}\|_0 \leq k. \tag{4.2}$$

Problem (4.2) can be reformulated as a MIO problem, by introducing a binary vector encoding the support of the regressor $\boldsymbol{w}$. The optimal cost for the equivalent

formulation is then given by

$$\begin{aligned} \min_{\boldsymbol{z}\in\{0,1\}^p:\sum_{j=1}^p z_j\leq k} \quad & \min_{\boldsymbol{w}\in\mathbb{R}^p} \quad \sum_{i=1}^n \ell(y_i, \boldsymbol{w}^T\boldsymbol{x}_i) + \frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2 \\ & \text{s.t.} \quad z_j = 0 \Rightarrow w_j = 0 \quad \forall j \in [p]. \end{aligned} \tag{4.3}$$

As we already noted in the introduction, exact MIO methods that solve Problem (4.3) scale up to $p \sim 10^5$. We next apply the backbone method to the sparse regression Problem (4.2).

During the backbone construction phase, we form the M subproblems, whose feature sets are denoted by $\mathcal{P}_m,\ m \in [M]$, using the `construct_subproblems` function (Algorithm 5). Within the m-th subproblem, $m \in [M]$, we may use a subset of data points $\mathcal{N}_m \subseteq [n]$ and different hyperparameter values k_m and γ_m. We form the backbone set as follows:

$$\begin{aligned} \mathcal{B} = \bigcup_{m=1}^M \Bigg\{ j : \quad & w_j^{(m)} \neq 0, \\ & \boldsymbol{w}^{(m)} = \underset{\substack{\boldsymbol{w}\in\mathbb{R}^p: \\ \|\boldsymbol{w}\|_0\leqslant k_m, \\ w_j=0\ \forall j\notin\mathcal{P}_m}}{\operatorname{argmin}} \sum_{i\in\mathcal{N}_m} \ell(y_i, \boldsymbol{w}^T\boldsymbol{x}_i) + \frac{1}{2\gamma_m}\|\boldsymbol{w}\|_2^2 \Bigg\}. \end{aligned} \tag{4.4}$$

We note that, instead of solving the sparse regression formulation in each subproblem, as shown in Equation (4.4), there is a possibility of solving a relaxation or a closely-related surrogate problem, such as the elastic net formulation [240]. We discuss this possibility in more detail in Section 4.3.1.

Finally, once the backbone construction phase is completed, we solve the original problem, considering only the features that are contained in the backbone set (i.e., exact solution computation in subset of backbone features), namely,

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathbb{R}^p} \quad & \sum_{i=1}^n \ell(y_i, \boldsymbol{w}^T\boldsymbol{x}_i) + \frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \|\boldsymbol{w}\|_0 \leqslant k, \\ & w_j = 0 \quad \forall j \notin \mathcal{B}. \end{aligned} \tag{4.5}$$

4.3.1 Implementation Details

In this section, we discuss some additional implementation aspects of the backbone method for the sparse regression problem.

Solving Subproblems via the Sparse Regression Formulation. Our first proposed approach to solve the subproblems relies on the actual sparse regression Formulation (4.2). In our implementation, we use the subgradient method of [50], which, besides fast, is especially strong in recovering the true support; it should be clear that any of the exact or heuristic solution methods discussed in Section 4.1.1 can be used. We empirically found this approach to be the most effective in terms of feature selection, namely, it achieves the lowest false detection rate within the backbone set.

A critical design choice in this approach concerns the *number of relevant features that should be extracted from each subproblem*. Unless our sampling procedure is perfectly accurate, it is possible that some subproblems will end up containing $k_{\text{s}} < k$ relevant features. If this is the case, we empirically observed that solution methods for Problem (4.2) quickly recover the k_{s} truly relevant features but struggle to select the remaining $k - k_{\text{s}}$, i.e., they spend a lot of time trying to decide which irrelevant features to pick. It is therefore crucial to either use cross validation within each subproblem (like the one we describe shortly), or apply an incremental selection procedure (e.g., forward stepwise selection). In our implementation, we use the following *incremental cross validation scheme for the subproblem support size*, which relies on progressively fitting less sparse models. We start at a small number of relevant features k_0, increment to $k_1 = k_0 + k_{\text{step}}$, $k_2 = k_0 + 2k_{\text{step}}$, and so forth. We stop after i steps if the improvement in terms of validation error of a model with $k_i + k_{\text{step}}$ and $k_i + 2k_{\text{step}}$ features is negligible compared to the error of a model with k_i features. Crucially, when training a model with $k_i > k_0$ features during cross validation, we use the best model with $k_i - k_{\text{step}}$ features as a warm-start.

Concerning the regularization parameter γ in Formulation (4.2), we apply a simple grid search (with grid length l) cross-validation scheme, between a small value, e.g.,

$\gamma_0 = \frac{p}{k \ n \ \max_i \|\boldsymbol{x}_i\|^2}$, and $\gamma_l = \frac{1}{\sqrt{n}}$, which is considered a good choice for most regression instances [78].

Solving Subproblems via Randomized Rounding. Instead of solving the MIO sparse regression Formulation (4.3), one can consider its boolean relaxation, whereby the integrality constraints $z_i \in \{0, 1\}$ are relaxed to $z_i \in [0, 1]$ [226]. The resulting solution $\hat{z}^{\text{rel}}$ will, in general, not be integral, so we propose to randomly round it by drawing $\hat{z}_i^{\text{bin}} \sim \text{Bernoulli}(\hat{z}_i^{\text{rel}})$. We form the backbone set by repeating the rounding procedure multiple times for each subproblem and collecting all features that had their associated binary decision variable set to 1 in at least one realization of the Bernoulli random vector. By doing so, we reduce the number of subproblems we solve and, instead, perform multiple random roundings per subproblem. As a result, the overall method is sped up, sacrificing, however, its effectiveness in terms of feature selection. Moreover, in this approach, we cannot directly control the number of relevant features extracted from each subproblem, so we heuristically keep the features that correspond to the $k_{\max}$ largest regressors. For these reasons, we did not use this approach in the computational results we present.

Solving Subproblems via Surrogate Formulations. In our second proposed approach to solve the subproblems, we formulate each subproblem using surrogate formulations, such as the lasso estimator [210] and its extensions. In particular, we utilize the elastic net formulation [240] to solve the m-th subproblem, $m \in [M]$, namely,

$$\boldsymbol{w}^{(m)} = \underset{\substack{\boldsymbol{w} \in \mathbb{R}^p: \\ w_j = 0 \ \forall j \notin \mathcal{P}_m}}{\text{argmin}} \quad \sum_{i \in \mathcal{N}_m \subseteq [n]} \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i) + \lambda_m \left[\mu_m \|\boldsymbol{w}\|_1 + \tfrac{1-\mu_m}{2} \|\boldsymbol{w}\|_2^2 \right]. \tag{4.6}$$

The popularity of ℓ_1-regularization is justified by the fact that it enjoys a number of properties that are particularly useful in practical applications. First and foremost, its primary mission is to robustify solutions against noise in the data and, specifically, against feature-wise perturbations. Second, the convexity of the ℓ_1-norm makes the

task of optimizing over it significantly easier. Third, ℓ_1-regularization provides sparser solutions than ℓ_2-regularization. We refer the interested reader to [30] and [228] for a detailed discussion on the equivalence of regularization and robustification.

We pick the hyperparameters of the elastic net formulation via cross validation. We select μ using grid search in the $[0, 1]$ interval. For each fixed μ, we perform grid search for $\lambda \in [\lambda_{\min}, \lambda_{\max}]$, where $\lambda_{\max}$ is the value of λ that leads to an empty model and $\lambda_{\min}$ is the value of λ that leads to a model consisting of $k_{\max}$ nonzero coefficients. Once the best elastic net hyperparameters are found, we refit and add to the backbone set those features whose associated coefficients are above a user-specified threshold.

Solving the Reduced Problem. After having formed the backbone set, we solve Problem (4.5) using the cutting planes method by [35]. As [50] observed, the computational time required for the MIO Formulation (4.3) to solve to provable optimality is highly dependent on γ; for smaller values of γ, problems with $p \sim 10^5$ can be solved in minutes, whereas, for larger values, it might take a huge amount of time to solve to provable optimality (although the optimal solution is usually attained fast). To address this issue, we impose a time limit on the cutting planes method for each value of γ during the cross validation process. In practice, we observe that the cutting planes method applied to the backbone set recovers the correct support in seconds (provided that the backbone set is sufficiently small).

4.3.2 Theoretical Justification

In this section, we present a theoretical result, which guarantees that, under certain conditions and with high probability, the true relevant features are selected in the backbone set. Our result, given in Theorem 3, theoretically justifies the proposed approach and, most importantly, as explained above, provides guidance for the selection of the hyperparameters of the method.

Model and Assumptions. The model that we consider is as follows. We have a random design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ such that each row $\boldsymbol{x}_i, i \in [n]$, is an iid copy of the

random vector $X = (X_1, ..., X_p) \sim \mathcal{N}(0, \boldsymbol{I}_p)$. We have fixed but unknown regressors $\boldsymbol{\beta} \in \{-1, 0, 1\}^p$ that satisfy $\|\boldsymbol{\beta}\|_0 \leq k$. Let $\mathcal{S}^{\text{true}}$ denote the set of indices that correspond to the support of the true regressor $\boldsymbol{\beta}$. The response vector is $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where the noise term $\boldsymbol{\varepsilon}$ consists of iid entries $\varepsilon_i \sim \mathcal{N}(0, \sigma^2), i \in [n]$. Thus, each entry in $\boldsymbol{Y}$ is distributed as $Y = X^\top \boldsymbol{\beta} + \varepsilon \sim \mathcal{N}(0, k + \sigma^2)$ (assuming that $\|\boldsymbol{\beta}\|_0 = k$). We further assume that $p > n \geq k$ and $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$. When we make asymptotic arguments, we take $p \to \infty$. We analyze a simplified version of the backbone method, which we outline in Section 5.8. Our goal is to show that, with high probability, $\mathcal{S}^{\text{true}} \subseteq \mathcal{B}$.

The model we examine is indeed very simple and rather unrealistic; nevertheless, such assumptions are standard for analyzing exact sparse regression methods (see, e.g., [114], [186], and [35]). Moreover, by the following result, we primarily aim to provide guidance for the selection of the hyperparameters of the method. We believe that the main contribution of the chapter is its practical relevance.

Main Result. We denote by ε_2 an upper bound on the probability that the `solve_subproblem` function fails to recover all relevant features that are included in the feature set of an arbitrary subproblem. We denote by $\mathcal{E}$ the event that the (simplified version of) backbone method fails, namely, $\exists j \in [p]$ such that $j \in \mathcal{S}^{\text{true}}$ and $j \notin \mathcal{B}$. We prove the following result:

Theorem 3. *Assume that the data is generated as discussed above, $p > n \geq k$, and $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$. Further, suppose that the sample size n satisfies $n \geq \theta(\sigma^2 + 2k)\log(\beta\alpha p)$, for all $\theta \geq 1$, so that $\varepsilon_2 = O(e^{-\theta}) \to 0$ as $p \to \infty$. Then, when $\alpha = O\left(\frac{n^{1-\phi}}{p}\right)$, for some $\phi < 1$, and when $M = O\left(\frac{\log(\alpha p k)}{\log\left(\frac{1}{1-\beta+\beta\varepsilon_2}\right)}\right)$, it holds that $P(\mathcal{E}) \to 0$ as $p \to \infty$, that is, the (simplified version of the) backbone method recovers with high probability the true relevant features in the backbone set.*

The proof is included in Section 5.8. This result asserts that the number of subproblems increases logarithmically with the product of the number of features that we sample from in each subproblem and the number of relevant features. Moreover, since β controls the subproblem size, the larger the subproblems are, the fewer subproblems

we need to solve. Finally, as ε_2 decreases and we therefore solve subproblems more accurately, we again need to solve fewer subproblems.

4.4 The Backbone of Decision Trees

Given data $\{(\boldsymbol{x}_i, y_i)\}_{i\in[n]}$, where, $\forall i \in [n]$, $\boldsymbol{x}_i \in \mathbb{R}^p$ (with $n \ll p$) and $y_i \in [K]$, decision trees recursively partition the feature space and assign a class label from $[K]$ to each partition. Formally, let T be a decision tree, $\mathcal{T}_B$ the set of branch nodes and $\mathcal{T}_L$ the set of leaf nodes. At each branch node $t \in \mathcal{T}_B$, a split of the form $\boldsymbol{a}_t^T \boldsymbol{x} < b_t$ is applied. Each leaf node $l \in \mathcal{T}_L$ is assigned a class label, typically via a majority vote among the class labels y_i of all data points that fall into leaf l after traversing the tree. The function $N(l; \boldsymbol{X})$ counts the number of data points in leaf l and $N_{\min}$ is a "minbucket" parameter that controls the minimum number of data points that are allowed to fall into any leaf. Finally, $g(T; \boldsymbol{X}, \boldsymbol{y}, \lambda) = \text{error}(T; \boldsymbol{X}, \boldsymbol{y}) + \lambda|T|$ is the objective and consists of two components. The first, $\text{error}(T; \boldsymbol{X}, \boldsymbol{y})$, is typically the misclassification error of the tree T on the training data $(\boldsymbol{X}, \boldsymbol{y})$. The second, $|T|$, is typically the number of branch nodes in the tree T. Then, at a high level, the decision tree problem can be stated as

$$T^\star = \text{argmin}_{T:\text{depth}(T)\leq D} \quad g(T; \boldsymbol{X}, \boldsymbol{y}, \lambda) \quad \text{s.t. } N(l; \boldsymbol{X}) \geq N_{\min} \; \forall l \in \mathcal{T}_L. \tag{4.7}$$

Ideally, we would like to exactly solve Problem (4.7) and therefore obtain a tree that achieves global optimality; in practice, however, this is still beyond the reach of MIO solvers. The optimal classification trees (OCT) framework provides a MIO formulation along with a set of heuristics that enable us to approximately solve Problem (4.7) in significantly larger dimensions than what was known. Nevertheless, despite the success of OCTs, the number of features p remains their primary bottleneck; currently, OCTs scale up to p in the 1,000s.

As we pointed in the introduction, the decision tree induction process has traditionally been addressed via scalable heuristic methods, such as CART. Nonetheless,

CART's training process has little to do with the actual misclassification objective and, as a result, the algorithm often settles with trees that are far from optimal for the original problem. The popularity of decision tree classifiers led to their wide use in ensemble models, such as bagging and boosting. Random forest, for example, combines multiple independently trained decision trees and typically boosts their performance, enjoying many of the benefits of ensemble learning, at the cost, however, of sacrificing interpretability.

The aforementioned limitations of decision tree-based methods, along with the fact that decision trees are indeed sparse models, are our main motivations in developing a backbone method for OCTs.

Relevant Features. Intuitively, each split in a decision tree is associated with a feature $j \in [p]$, so there are at most $2^D - 1$ relevant features in tree of max depth D. We define that feature j relevant if at least one split performed on it. During the backbone construction phase, we again form M distinct and tractable subproblems.

Constructing Subproblems. Similarly to regression, in the m-th subproblem, $m \in [M]$, we only consider the features included in the subproblem's feature set $\mathcal{P}_m$ (constructed using the `construct_subproblems` function of Section 4.2.3) and, possibly, randomly sample a subset of data points $\mathcal{N}_m \subseteq [n]$. Note that, in formulating each subproblem, we may use different hyperparameter values $\lambda_m, N_{\min,m}$ and D_m.

Forming the Backbone Set. Let us denote by $\mathcal{T}(\boldsymbol{X}, \boldsymbol{y}, D, N_{\min})$ the set of all feasible trees on input data $(\boldsymbol{X}, \boldsymbol{y})$ and by $\boldsymbol{a_t}$ the split performed at branch node t. Then, the backbone set can be written as the union of the solutions to all subproblems, namely,

$$\mathcal{B} = \bigcup_{m=1}^{M} \Bigg\{ j : \quad \sum_{t \in \mathcal{T}_B^{(m)}} a_{tj}^{(m)} \geq 1, \\ T^{(m)} = \underset{\substack{T \in \mathcal{T}(\boldsymbol{X}_{\mathcal{N}_m, \mathcal{P}_m}, \\ \boldsymbol{y}_{\mathcal{N}_m}, D_m, N_{\min,m})}}{\operatorname{argmin}} \quad g(T; \boldsymbol{X}_{\mathcal{N}_m, \mathcal{P}_m}, \boldsymbol{y}_{\mathcal{N}_m}, \lambda_m) \Bigg\}. \tag{4.8}$$

Importantly, in solving each of the subproblems in (4.8), we need not solve the OCT formulation, shown in (4.7); instead, we propose solving each subproblem using scalable heuristic methods, such as CART. In fact, we empirically found that applying the OCT framework to subproblems does not significantly improve the backbone accuracy (i.e., fraction of relevant features included in the backbone set). We use cross validation within each subproblem $m \in [M]$ to tune the hyperparameters $D_m, N_{\min,m}, \lambda_m$.

Solving the Reduced Problem. Once the backbone set $\mathcal{B}$ is constructed, we solve the OCT Formulation (4.7), considering only the features in $\mathcal{B}$, i.e.,

$$T^\star = \text{argmin}_{T:\text{depth}(T)\leq D} \quad g(T; \boldsymbol{X}_\mathcal{B}, \boldsymbol{y}, \lambda) \quad \text{s.t. } N(l; \boldsymbol{X}_\mathcal{B}) \geq N_{\min} \; \forall l \in \mathcal{T}_L. \tag{4.9}$$

Extension to Optimal Classification Trees with Hyperplane Splits. Instead of limiting the tree learning method to univariate splits, where, $\boldsymbol{a_t} \in \{0,1\}^p$ for any branch node t, multivariate splits can also be used. If this is the case, it is important that the number of features that participate in each split is artificially constrained (which translates to a sparsity constraint on $\boldsymbol{a_t}$), otherwise the backbone set will generally not be small enough to substantially reduce the problem dimension.

Connections with the Random Subspace Method and Random Forest. Feature bagging methods have had significant success when applied to ensembles of decision trees. The random subspace method [135] relies on training each decision tree in the ensemble on a random subset of the features instead of the entire feature set. In random forest [62], a subset of the features is considered in each split of each decision tree. Our backbone method for OCTs relies on the `construct_subproblems` function, so each tree induced during the backbone construction phase is also trained on a subset of features. Thus, our approach enjoys many of the benefits of feature bagging (e.g., parallelizeability), while, at the same time, its output is a single, interpretable tree.

4.5 Experiments on Synthetic Data

In this section, we investigate the performance of the backbone method on synthetic datasets generated according to ground truth models that are known to be sparse. We start by describing the data generating methodology, the metrics, and the algorithms that we use throughout this (as well as the following) section. Then, we explore the scalability and performance of the method, as well as the sensitivity to the method's hyperparameters and components. Our primary goal in this section is to shed light on the behavior of the proposed backbone method and not to benchmark the backbone method compared to state-of-the-art alternatives; this is, in fact, the focus of Section 4.6.

4.5.1 Data Generating Methodology

In all our synthetic experiments throughout this section, we generate data according to the following methodology.

Design Matrix. We assume that the *input data* $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ are i.i.d. realizations from a p-dimensional zero-mean normal distribution with covariance matrix $\boldsymbol{\Sigma}$, i.e., $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{0}_p, \boldsymbol{\Sigma}), i \in [n]$. The covariance matrix $\boldsymbol{\Sigma}$ is parameterized by the correlation coefficient $\rho \in [0, 1)$ as $\Sigma_{ij} = \rho^{|i-j|}, \forall i, j \in [p]$. As $\rho \to 1$, the columns of the data matrix $\boldsymbol{X}$, i.e., the features, become more alike which should impede the discovery of nonzero components of the true regressor $\boldsymbol{w}_{\text{true}}$ by obfuscating them with highly correlated look-alikes. In our experiments, we focus on high correlation regimes (e.g., $\rho = 0.6$ or even $\rho = 0.9$).

Sparse Linear Regression Data. For linear regression, the unobserved true regressor $\boldsymbol{w}_{\text{true}}$ is constructed at the beginning of the process and has exactly k-nonzero components at indices selected uniformly without replacement from $[p]$. Likewise, the nonzero coefficients $\boldsymbol{w}_{\text{true}}$ are drawn uniformly at random from the set $\{-1, +1\}$. We next generate the *response vector* $\boldsymbol{y}$, which satisfies the linear relationship $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}_{\text{true}} + \boldsymbol{\varepsilon}$, where $\varepsilon_i, i \in [n]$, are i.i.d. noise components from a normal distribution,

scaled according to a chosen signal-to-noise ratio $\sqrt{\text{SNR}} = \|\boldsymbol{X}\boldsymbol{w}_{\text{true}}\|_2/\|\boldsymbol{\varepsilon}\|_2$. Evidently as the SNR increases, recovery of the unobserved true regressor $\boldsymbol{w}_{\text{true}}$ from the noisy observations can be done with higher precision.

Sparse Logistic Regression Data. For logistic regression, the true regressor is constructed in the exact same manner. The signal $\boldsymbol{y}$ is computed according to

$$\boldsymbol{y} = \text{sign}(\boldsymbol{X}\boldsymbol{w}_{\text{true}} + \boldsymbol{\varepsilon}).$$

Classification Tree Data. For classification trees, we first create a full binary tree T_{true} (*ground truth tree*) of given depth D. The structure of T_{true} is determined as follows.

- *Relevant features:* We randomly pick k relevant features among the entire feature set. At each split node, we select a relevant feature to split on. To ensure that all relevant features are actually informative, we require that each of them appears in at least r split nodes in the tree. Thus, the number of relevant features k must satisfy $k \leq \frac{2^D-1}{r}$.

- *Split thresholds:* Within each split node, we randomly pick a split threshold, taking care to maintain consistency of feature ranges across paths in the tree. Moreover, let $x \in [x_{\min}, x_{\max}]$ be the feature on which we split at node t. To ensure that splits are reasonably balanced, we require that the split threshold $b_t \in [x_{\min} + \frac{x_{\max}-x_{\min}}{2} \cdot f,\ x_{\max} - \frac{x_{\max}-x_{\min}}{2} \cdot f]$, where $f \in [0, 1]$ is the parameter that determines how balanced the splits are. In our experiments, we use $f = \frac{1}{2}$.

- *Labels:* We assign class labels to leaf nodes in such a way that no sibling leaves correspond to the same class. We denote by $\mathcal{C}$ the set of all class labels and by $c_l \in \mathcal{C}$ the class label of leaf l. Furthermore, let $K = |\mathcal{C}|$ the total number of classes. In our experiments, we use $K = 2$ and hence examine binary classification problems.

Next, we generate data from T_{true} by setting, for each $i \in [n]$, $y_i = c_l$, where l is the leaf where data point $\boldsymbol{x}_i$ falls after traversing the tree.

4.5.2 Metrics

In our computational study in this section and in Section 4.6, we evaluate the quality of each method based on the following metrics:

- *Support recovery accuracy* (SR-ACC): Measures the fraction of the k relevant features that were actually selected by the estimator. For example, in the context of regression, we have

$$\frac{|\{j : w_j \neq 0,\ w_{\text{true},j} \neq 0\}|}{k}.$$

- *Support recovery false alarm rate* (SR-FA): Measures the ratio of number irrelevant features selected over total number of features selected.

- *Fraction of features used that are relevant:* Measures the ratio of the number of relevant features used over the total number of features used in the learned model. This metric simultaneously captures support recovery accuracy and model simplicity, and is particularly useful in decision tree models, whereby one feature might be part of the ground truth tree and yet have little impact on the classification task.

- *Prediction accuracy* (R^2 or AUC): Evaluates the out-of-sample performance of the estimator. We use the R^2 statistic for linear regression and the area under the curve (AUC) for logistic regression and for classification trees (since we deal with binary classification problems).

- *Optimality gap* (OG): Measures the gap between the lower and upper objective bound during the solution process of an MIO problem and, specifically, of Problem (4.3).

- *Learned tree depth:* Reports the depth of the learned decision tree model.

- *Computational time* (T): Total amount of time used (in seconds).

Additionally, to assess the quality of each component of the backbone method, we also record the following statistics:

- *Backbone accuracy*: Measures the fraction of the k relevant features that were selected in the backbone set.

- *Backbone size*: Number of features included in the backbone set.

- *Subproblem feature sets accuracy*: Measures the fraction of the k relevant features that were sampled in at least one subproblem's feature set.

All experimental results were obtained over 10 independently generated datasets. In each experiment, we report both the mean and standard deviation of each metric. All out-of-sample metrics were obtained from independently generated test sets of size $n_{\text{test}} = 2,000$.

4.5.3 Algorithms & Software

In this section, we summarize the algorithms and software that we use in our experiments. For the sparse regression problem, we use the following algorithms:

- `SR`: Implementation of the cutting planes method by [35] in Julia using the commercial MIO solver Gurobi [123]. Uses the subgradient method [50] to compute a warm start. Solves the sparse regression formulation to optimality or near-optimality.

- `SR-REL`: Implementation of the subgradient method by [50] in Julia using the commercial Interpretable AI software package [139]. Solves the sparse regression heuristically by considering its boolean relaxation and iteratively alternating between a sub-gradient ascent step and a projection step.

- `ENET`: `glmnet` Fortran implementation [110], using the Julia wrapper. Solves the elastic net formulation using cyclic coordinate descent.

For the decision tree problem, we use the following algorithms:

- `OCT`: Implementation of the OCT-learning method from the OT framework [45] in Julia using the commercial Interpretable AI software package [139]. Solves the decision tree formulation via a tailored local search procedure.

- `CART`: Implementation of the CART heuristic [63] using the Julia wrapper for the scikit-learn package [183].

- `RF`: Implementation of the random forest classifier [62] using the Julia wrapper for the scikit-learn package [183].

To address high-dimensional regimes, we consider the following methods that incorporate a feature selection component:

- `SIS-ENET`: Implementation of the sure independence screening [100] feature selection heuristic in Julia, as per Section 4.2.2, followed by `ENET` on the reduced feature set.

- `RFE`: Implementation of the popular recursive feature elimination algorithm [126] using a Julia wrapper for the scikit-learn package [183]. For sparse regression, we use linear regression to eliminate features, with a step of 100 features; then, we apply `SR` on the selected features (tuned using cross-validation). For classification trees, we use `CART` to eliminate features, with a step of 100 features; then, we apply `CART` on the selected features (tuned using cross-validation).

- `DECO`: Implementation of the DECO framework by [219] in Julia. We partition the feature space into 5 subsets and, after the de-correlation step, we perform feature selection in each subset using lasso. Then, we apply `ENET` on the selected features (tuned using cross-validation).

Finally, our proposed backbone method is implemented as outlined below:

- `BB`: Implementation of the backbone method in Julia. The components and parameterization of the method are discussed in each experiment separately.

We remark that, in the results that we present, we did not make an effort to fine-tune the method's parameters; instead, we selected them based on Theorem 3 and on empirical evidence.

All experiments were performed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running CentOS release 7. Moreover, all methods' hyperparameters are tuned using the holdout method for cross validation, whereby we split the training set into actual training and validation data at a 0.7 ratio.

4.5.4 Scalability with the Number of Features

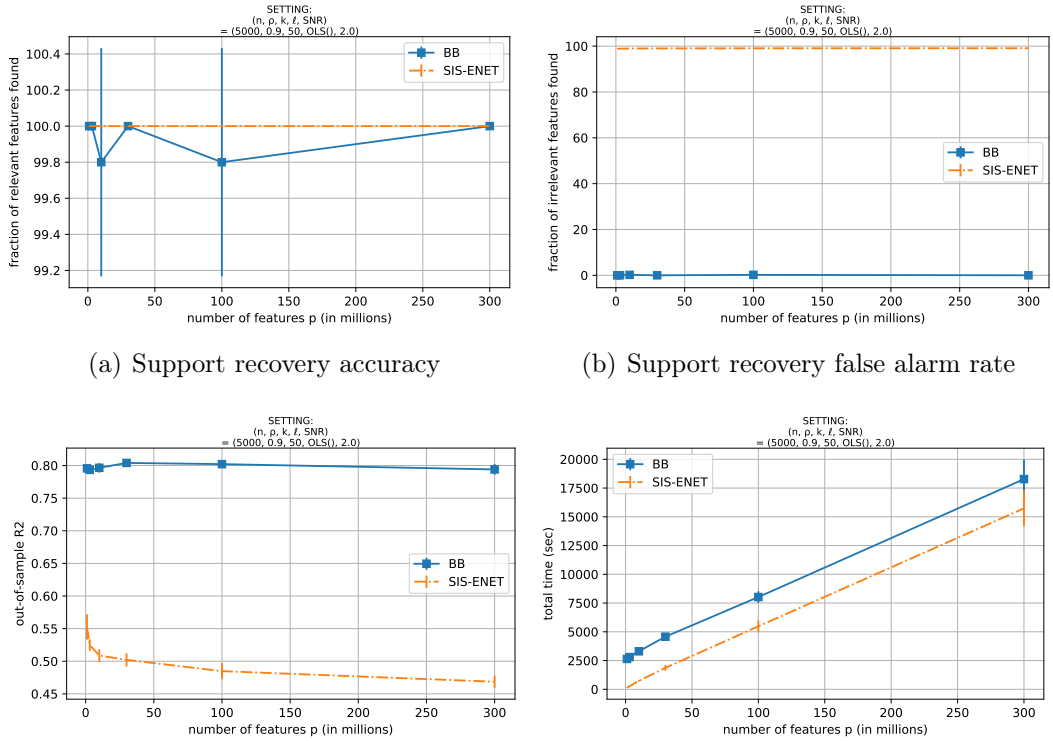
In this experiment, we examine the scalability of `BB` as the number of features p increases. We show that `BB` accurately scales to ultra-high dimensional problems and notably outperforms baseline heuristics.

Sparse Linear Regression. We consider a sparse linear regression problem with $n = 5,000$ data points, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of features $p \in \{10^6, 3 \cdot 10^6, \ldots, 3 \cdot 10^8\}$.

We tune `BB` as follows. We select the `screen` function's parameter α such that all but $4 \cdot n = 20,000$ features are eliminated. We set $\beta = 0.5$ and solve $M = 10$ subproblems. We set $B_{\max} = 250$. We solve the subproblems using `SR-REL`; in the m-th subproblem, we cross-validate 3 values for $k_m \in \{\lceil \frac{k}{3} \rceil, \lceil \frac{2k}{3} \rceil, k\}$ and set $\gamma_m = \frac{1}{\sqrt{n_m}}$. We solve the reduced problem using `SR` with a time limit of 5 minutes; we cross-validate 5 values for the hyperparameter γ. As a baseline, we compare `BB` with `SIS-ENET`, whereby we select αp features using `SIS` and then apply `ENET`. We tune `ENET` as described in Section 4.3.1 and, specifically, we cross-validate 5 values for the hyperparameter $\mu \in [0, 1]$ (the pure lasso model is included in the cross-validation procedure). We discard from the final model any feature whose corresponding regressor has magnitude $\leq 10^{-6}$.

Figure 4-1 presents the results for this experiment. `BB` achieves near-perfect accuracy for problems with up to 300 million features and substantially outperforms `SIS-ENET`, in terms of both support recovery accuracy (in that `BB` recovers almost

the entire true support with near zero false positives, whereas `SIS-ENET` recovers the true support at the cost of a large number of false positives) and out-of-sample predictive performance. As far as the computational time is concerned, we observe that the overhead of `BB` over `SIS-ENET` is by no means prohibitive; we are able to solve problems with 10 million features in less than an hour and problems with 300 million features in less than 10 hours.



(a) Support recovery accuracy

(b) Support recovery false alarm rate

(c) Out-of-sample R^2

(d) Computational Time (sec)

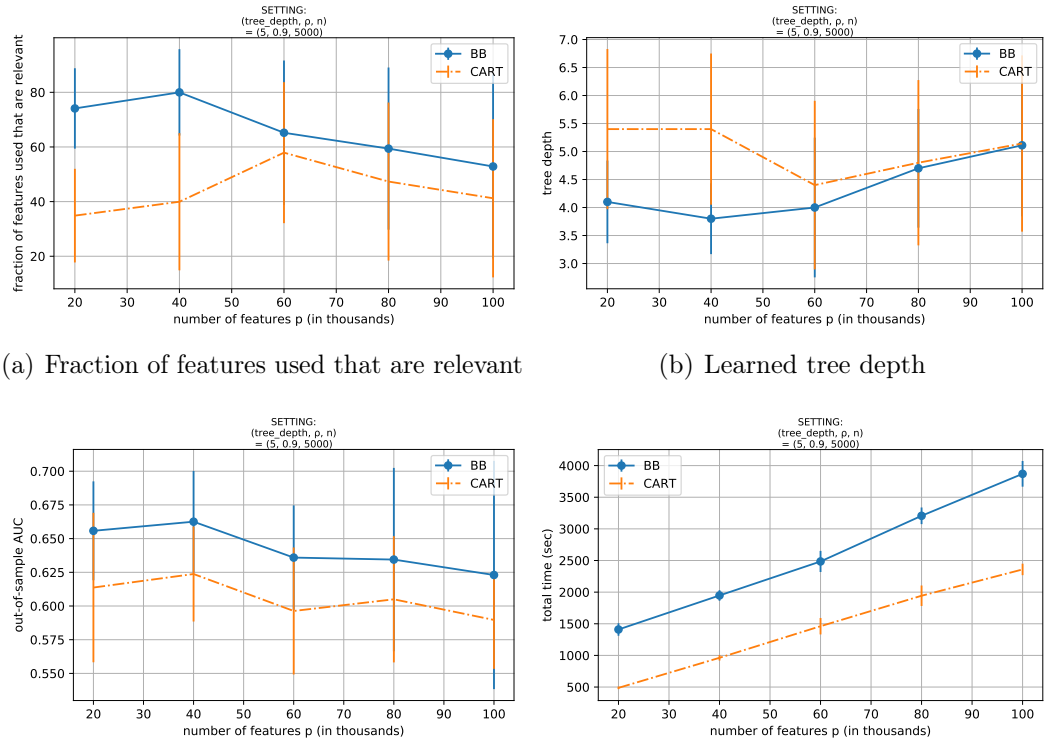
Figure 4-1: Scalability with the number of features for sparse linear regression.

Classification Trees. We consider a classification tree problem with $n = 5,000$ data points, tree depth of $D = 5$ and $k = 31$ relevant features, and correlation $\rho = 0.9$. We vary the number of features $p \in \{2 \cdot 10^4, 4 \cdot 10^4, \ldots, 10^5\}$.

We tune `BB` as follows. We select the `screen` function's parameter $\alpha = 1$ such that no features are eliminated. We set $\beta = 0.1$ and solve $M = 50$ subproblems. We set $B_{\max} = 250$. We solve the subproblems using `CART`; in the m-th subproblem, we cross-validate $D_m \in \{2, \ldots, D-2\}$ (we also cross-validate the minbucket and

complexity parameter of CART). We solve the reduced problem using OCT; we cross-validate $D \in \{3, \ldots, D+2\}$ (we also cross-validate the minbucket and complexity parameter of OCT). As a baseline, we compare BB with CART, tuned in the exact same way as OCT in solving the reduced problem for BB.

As can be observed in Figure 4-2, BB outperforms CART in terms of out-of-sample predictive performance for problems with up to $100,000$ features. Moreover, among the features that are used in the split nodes of the learned classification tree, BB selects a substantially higher fraction of relevant ones and, at the same time, the learned tree is simpler (i.e., of smaller depth). This particular configuration of BB solves problems with $100,000$ features in approximately an hour and the computational time scales linearly with the number of features.



(a) Fraction of features used that are relevant

(b) Learned tree depth

(c) Out-of-sample AUC

(d) Computational Time (sec)

Figure 4-2: Scalability with the number of features for classification trees.

4.5.5 Scalability with the Number of Samples

In this experiment, we examine the scalability of `BB` as the number of samples n increases. We show that `BB` outperforms baseline heuristics in the context of sparse linear regression, sparse logistic regression, and classification trees.

Sparse Linear Regression. We consider a sparse linear regression problem with $p = 10^6$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{1,000, 3,000, \ldots, 9,000\}$.

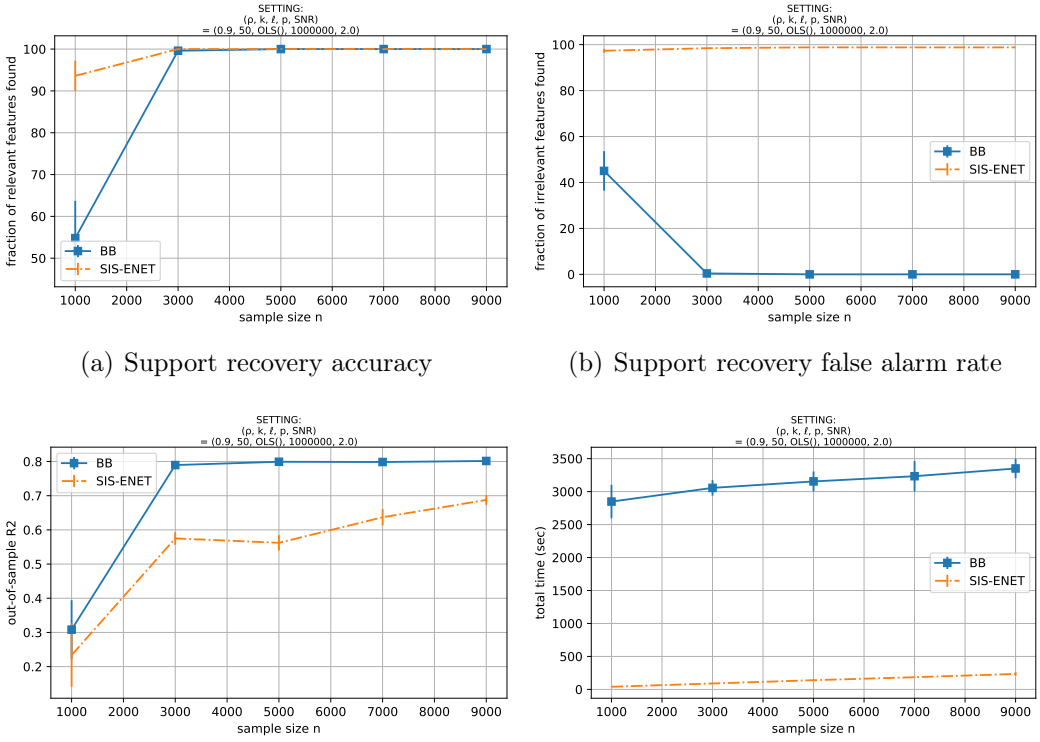
We tune `BB` and `SIS-ENET` as described in the first experiment in Section 4.5.4, under the following modifications. For `BB`, we now select the `screen` function's parameter α such that all but $10,000$ features are eliminated, $\beta = 0.5$ and solve $M = 15$ subproblems.

Figure 4-3 presents the results for this experiment. `BB` outperforms `SIS-ENET` as the number of samples increases, in terms of both support recovery accuracy and out-of-sample predictive performance, and solves problems with 1 million features and $9,000$ samples in less than an hour.

Sparse Logistic Regression. We consider a sparse logistic regression problem with $p = 10^6$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{3,000, 5,000, \ldots, 11,000\}$.

We tune `BB` and `SIS-ENET` as described in the sparse linear regression experiment preceding this one (Section 4.5.5), under the following modifications. For `BB`, we now cross-validate the hyperparameter γ_m within each subproblem and increase the time limit for the reduced problem to 15 minutes.

In Figure 4-4, we show that `BB` outperforms `SIS-ENET` as the number of samples increases, in terms of both support recovery accuracy and out-of-sample predictive performance, and solves sparse logistic regression problems with 1 million features and $11,000$ samples in less than five hours. This experiment illustrates that `BB` performs equally well in classification problems, whereby the logistic loss is used instead of the least squares loss.



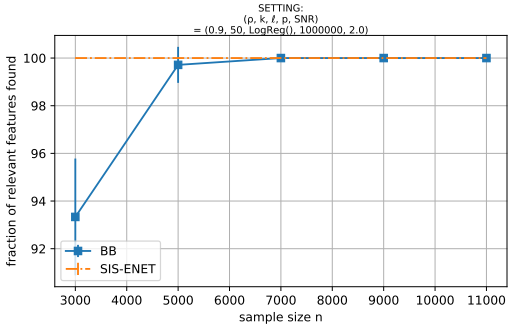
(a) Support recovery accuracy

(b) Support recovery false alarm rate

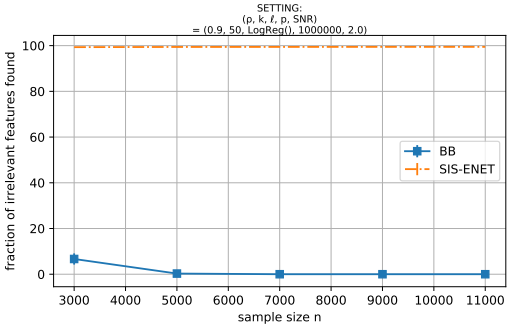
(c) Out-of-sample R^2

(d) Computational Time (sec)

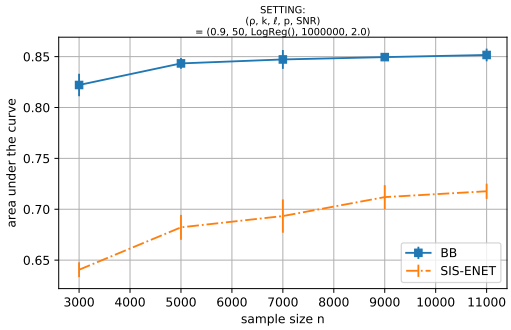
Figure 4-3: Scalability with the number of samples for sparse linear regression.



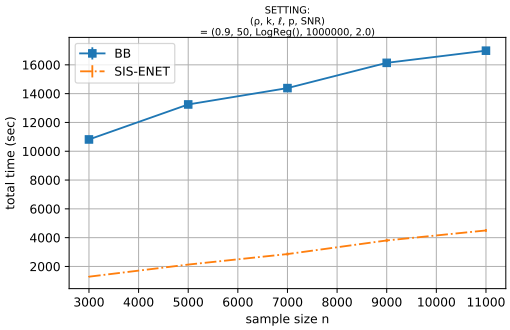
(a) Support recovery accuracy



(b) Support recovery false alarm rate



(c) Out-of-sample AUC



(d) Computational Time (sec)

Figure 4-4: Scalability with the number of samples for sparse logistic regression.

Classification Trees. We consider a classification tree problem with $p = 100,000$ features, tree depth of $D = 3$ and $k = 7$ relevant features, and correlation $\rho = 0.6$. By considering a much simpler ground truth tree than in Section 4.5.4, the methods under investigation will hopefully be able to learn a tree that is closer to the truth. We vary the number of data points $n \in \{3,000, 5,000, 7,000\}$.

We tune `BB` and `CART` as described in the classification tree experiment in Section 4.5.4, under the following modifications. For `BB`, we now select $\beta = 0.5$ and solve $M = 10$ subproblems (i.e., we solve fewer, larger subproblems compared to Section 4.5.4).

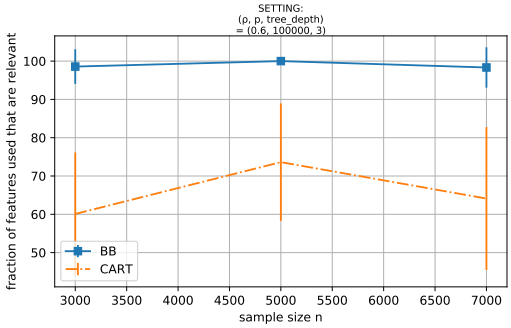
The results are presented in Figure 4-5. Both methods achieve near perfect out-of-sample AUC and `BB` is computationally more intensive; this is likely due to the fact that `OCT`, which is used to solve the reduced problem, is more sensitive to the number of samples in the data. Nevertheless, `BB` results in trees that are much simpler and much closer to the ground truth, in that the fraction of features used that are relevant is close to 100% and the learned tree's depth is, on average, within 1 of the ground truth tree's depth.

4.5.6 Comparison with Exact Methods Applied to the Entire Feature Set

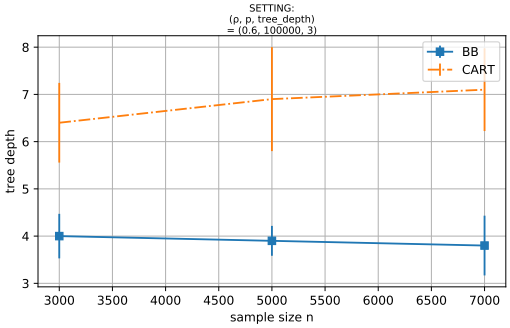
In this experiment, we compare `BB` with `SR` or `OCT` applied to the entire feature set in problems that are sufficiently small and `SR` or `OCT` scale. We show that, in such regimes, `BB` competes with optimal or near-optimal solutions, while substantially reducing the computational time and/or MIO optimality gap.

Sparse Linear Regression. We consider a sparse linear regression problem with $p = 20,000$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{1,000, 2,000, \ldots, 5,000\}$.

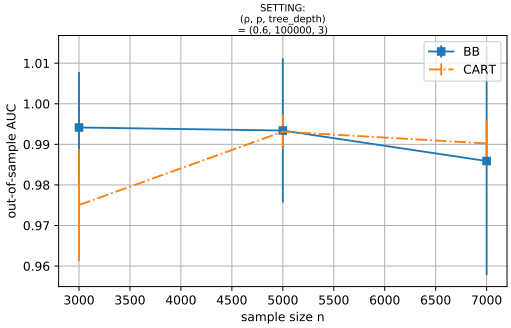
We tune `BB` as described in the first experiment in Section 4.5.4, under the following modifications. For `BB`, we now select the `screen` function's parameter α such that all but $5,000$ features are eliminated, we set $\beta = 0.4$, and solve $M = 10$ subproblems.



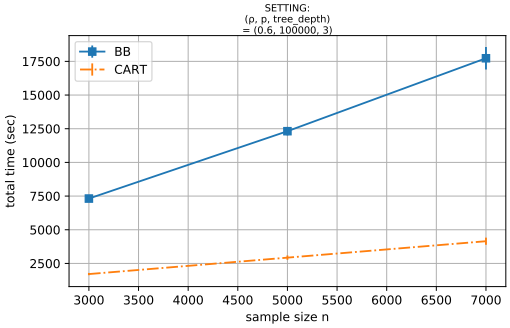
(a) Fraction of features used that are relevant



(b) Learned tree depth



(c) Out-of-sample AUC

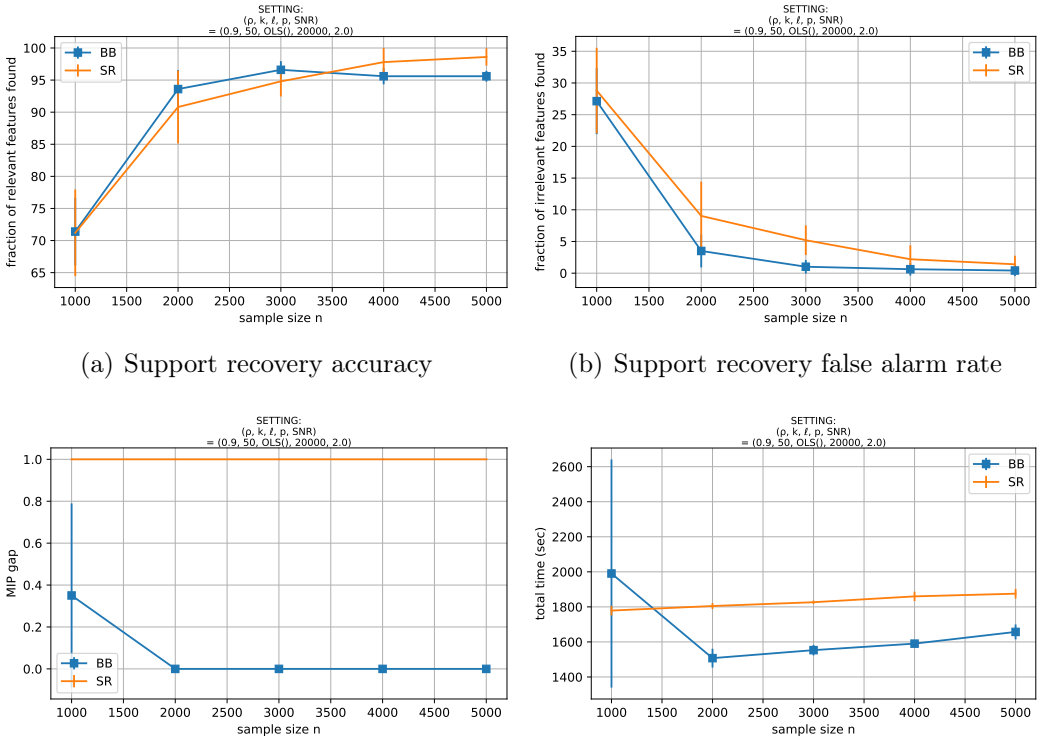


(d) Computational Time (sec)

Figure 4-5: Scalability with the number of samples for classification trees.

We compare BB with SR applied to the entire feature set and tuned exactly as when solving the reduced problem in BB.

Figure 4-6 reports the support recovery accuracy, MIO optimality gap, and computational time of each method as function of the sample size n. Both methods achieve near perfect support recovery accuracy at similar rates; however, BB seems to have an edge when it comes to support recovery false alarm rate. One possible explanation for this is that the first phase in BB eliminates irrelevant features that SR ends up selecting. Additionally, while the solution returned by SR comes with no optimality guarantee, the optimality gap for BB quickly drops to 0 as the number of samples increases, albeit for the reduced problem. In terms of computational time, BB indeed performs effective feature selection and enables us to solve the reduced problem's sparse regression MIO formulation to near-optimality faster.



(a) Support recovery accuracy

(b) Support recovery false alarm rate

(c) Optimality Gap

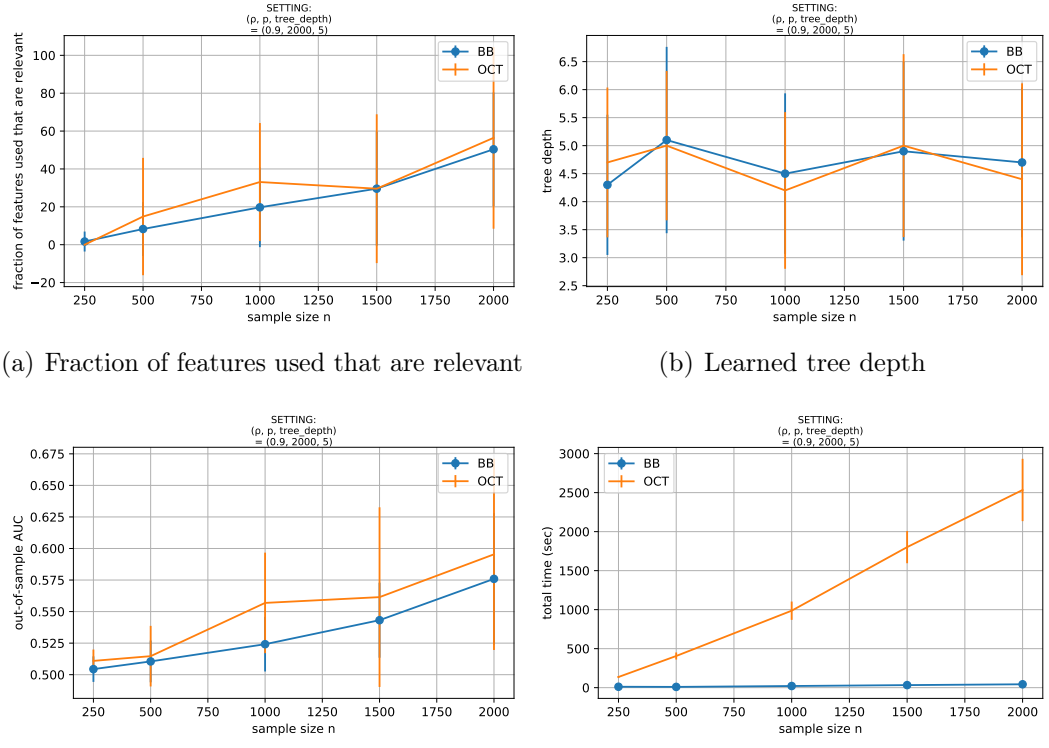
(d) Computational Time (sec)

Figure 4-6: Comparison with sparse linear regression applied to the entire feature set.

Classification Trees. We consider a classification tree problem with $p = 2,000$ features, tree depth of $D = 5$ and $k = 31$ relevant features, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{250, 500, 1,000, 1,500, 2,000\}$.

We tune `BB` as described in the classification tree experiment in Section 4.5.4, under the following modifications. Since our focus now is solely on feature selection, we incorporate the screening step into `BB`. Thus, we select $\alpha = 0.5$, we screen features and construct subproblems using the logistic loss, we set $\beta = 0.5$, and solve $M = 10$ subproblems.

We present the results in Figure 4-7. In terms of support recovery and structure of the learned tree, the two methods performs similarly. Although `OCT` has a slight edge in terms of predictive power, the gains of `BB` in terms of computational time are tremendous.



(a) Fraction of features used that are relevant

(b) Learned tree depth

(c) Out-of-sample AUC

(d) Computational Time (sec)

Figure 4-7: Comparison with optimal classification trees applied to the entire feature set.

4.5.7 Sensitivity to the Backbone Method's Hyperparameters and Components

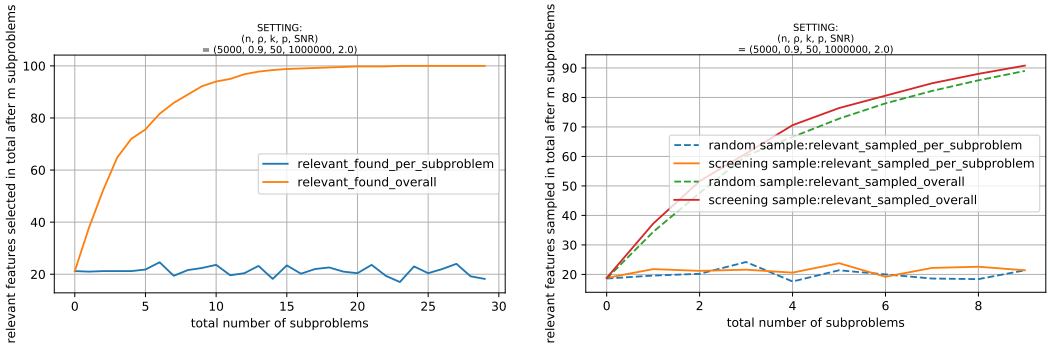
In the remainder of this section, we present a detailed analysis on the sensitivity of `BB` to its components and hyperparameters in the context of regression. The sensitivity analysis for the decision tree problem leads to near-identical conclusions, so we do not include it in the chapter.

We consider a sparse linear regression problem with $n = 5,000$ data points, $p = 10^6$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. Unless stated otherwise, we set the parameters of `BB` as follows: we use the `screen` function described in Algorithm 4 and set $\alpha = 0.01$ so that all but 10^4 features are eliminated; we construct subproblems using the `construct_subproblems` function (Algorithm 5) with $M = 10$ and $\beta = 0.2$; we impose a maximum allowable backbone size of $B_{\max} = 1,000$ features; we solve the subproblems using `SR-REL` and the reduced problem using `SR` with a time limit of 5 minutes.

Number of Subproblems. In this experiment, we study the backbone accuracy (i.e., fraction of relevant features that are included in the backbone set) as function of the subproblem number $m \in [M] = [30]$, that is, we report the accuracy for the same run of `BB`, after solving the first subproblem, after solving the second subproblem, and so forth. The results are presented in Figure 4-8(a). We make the following remarks:

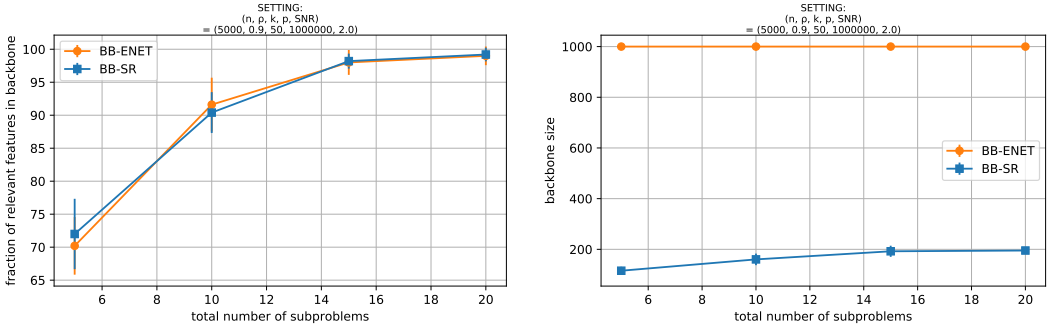
- The blue line corresponds to the fraction of relevant features that are selected (i.e., added to the backbone set) in the m-th subproblem, as function of the subproblem number m. Since the m-th subproblem's feature set consists of a fraction $\beta = 0.2$ of the total number $\alpha p = 10^4$ of features, we would expect to select a fraction β of the relevant features in the m-th subproblem (assuming that we solve subproblems perfectly). We observe that, on average, the fraction of relevant features selected in the m-th subproblem is slightly higher; this is due to the more informed sampling scheme used in the `construct_subproblems` function (Algorithm 5).

- The orange line corresponds to the fraction of relevant features that have been selected overall (i.e., across all m subproblems) after solving the m-th subproblem, as function of the subproblem number m. With all other backbone parameters being fixed, as the number of subproblems increases, the backbone accuracy increases and stabilizes after few subproblems, at the expense of an increased computational time. In this case, the backbone set achieves perfect accuracy after $M = 15$ subproblems.



(a) Number of Subproblems: Fraction of relevant features found in each subproblem and in total

(b) Function `construct_subproblems`: Fraction of relevant features sampled in each subproblem and in total



(c) Function `solve_subproblem`: Backbone accuracy

(d) Function `solve_subproblem`: Backbone size

Figure 4-8: Hyperparameter M and components of the backbone method.

Function `construct_subproblems`**.** In this experiment, we examine the impact of the `construct_subproblems` function on `BB`. We compare two approaches for the `construct_subproblems` function. The first one is based on constructing subproblems' feature sets by sampling features uniformly at random, as per the theoretical analysis

in Section 4.3.2 and in Section 5.8; we call this approach `random_sample`. The second one is the non-uniformly random sampling approach described in Section 4.2.3; we call this approach `screening_sample`. The results are presented in Figure 4-8(b). The dashed lines correspond to the fraction of relevant features that are sampled in the m-th subproblem's feature set and overall after m subproblems (across all m subproblems), as function of the subproblem number m, under the `random_sample` approach; the solid lines correspond to the `screening_sample` approach. With all other backbone parameters being fixed, we observe the following:

- The `screening_sample` approach samples a slightly larger number of relevant features in the feature set of each subproblem.

- The `screening_sample` approach samples a notably larger number of relevant features across all subproblems. This is due to the fact that, when the `screening_sample` approach does not sample a relevant feature in a subproblem's feature set, it is more likely to sample another relevant feature in its place; this is not the case for the `random_sample` approach.

Function `solve_subproblem`**.** In this experiment, we explore the impact of the `solve_subproblem` function on `BB`. We compare two approaches for the `solve_subproblem` function: in the first approach, we use `SR-REL` to solve subproblems, whereas in the second, we use `ENET` (we set $\mu = 1$ so that the pure lasso, which generally leads to sparser models compared to the elastic net, is used). With all other backbone parameters being fixed, we observe the following:

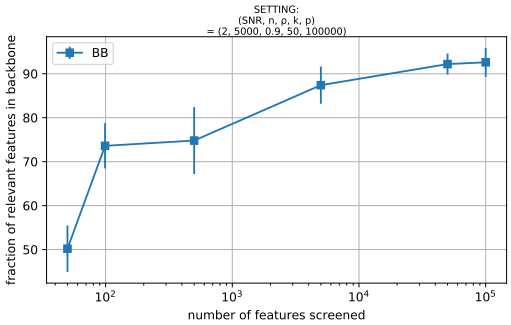
- The two approaches perform comparably in terms of the backbone accuracy (Figure 4-8(c)).

- The `ENET`-based approach results in a large backbone set, which, in fact, exceeds the limit of $B_{\max} = 1,000$ features. We conclude that this approach does not produce sufficiently sparse solutions to the subproblems (Figure 4-8(d)).

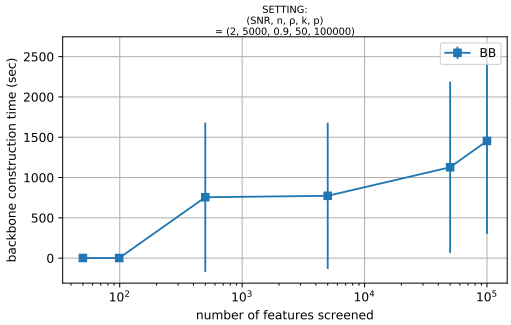
Number of Features Eliminated by `screen`. In this experiment, we test the impact on the backbone set of the hyperparameter α, which determines what fraction of the features will be eliminated by the `screen` function. Figures 4-9(a) and 4-9(b) indicate that a smaller value of α (which results in fewer features surviving the screening step) is beneficial, provided that no relevant features are missed during this step. With all other backbone parameters being fixed, as α increases, it becomes more likely to miss a relevant feature from the backbone set and it takes more to solve the subproblems (as subproblems are both bigger and harder).

Number of Features per Subproblem. In this experiment, we test the impact on the backbone set of the subproblem size, which we control through the hyperparameter β. Figures 4-9(c) and 4-9(d) indicate that a value $\beta \approx 0.5$ is ideal for the problems that we consider. With all other backbone parameters being fixed, as β increases, we observe that the computational time slightly increases, except when the subproblem size is too small, and the backbone accuracy also increases. Intuitively, we want β to be large enough, so that enough signal is contained in the subproblems, and small enough, so that the subproblems can be solved fast.

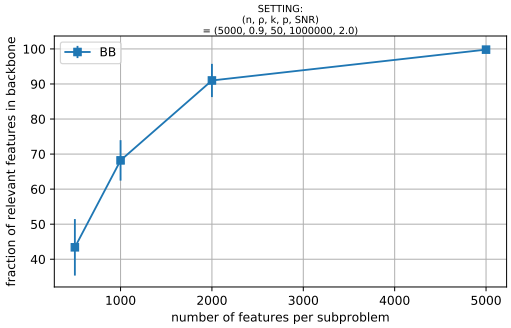
Maximum Backbone Size. In this experiment, we explore the impact on the backbone set of the maximum backbone size, controlled by the hyperparameter $B_{\max}$. Figures 4-9(e) and 4-9(f) indicate that, with all other backbone parameters being fixed, as $B_{\max}$ increases, the backbone accuracy slightly increases; the computational time is not affected, except when the maximum backbone size is too small. When this is the case, the reduced problem can be solved very fast and therefore the overall computational time drops; this, however, comes with a higher risk of not including relevant features in the backbone set. We also remark that the maximum backbone size and number of iterations of the hierarchical backbone algorithm (Algorithm 3) are connected, since decreasing the maximum backbone size will likely result in more iterations.



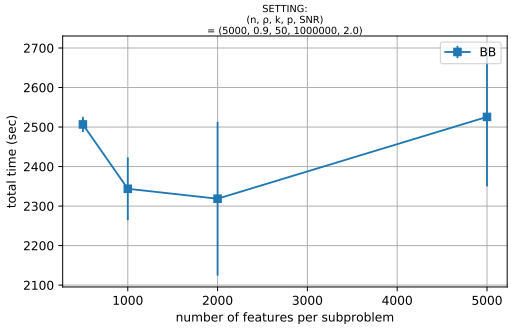
(a) Number of Features Eliminated by `screen`: Backbone accuracy



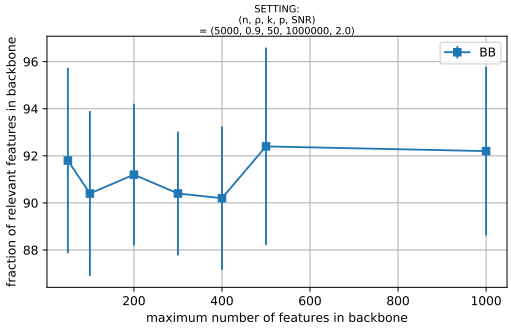
(b) Number of Features Eliminated by `screen`: Computational time (sec)



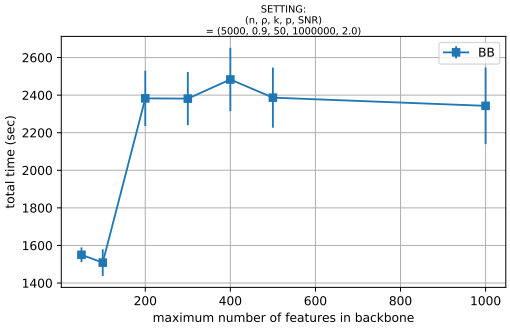
(c) Number of Features per Subproblem: Backbone accuracy



(d) Number of Features per Subproblem: Computational time (sec)



(e) Maximum Backbone Size: Backbone accuracy



(f) Maximum Backbone Size: Computational time (sec)

Figure 4-9: Hyperparameters α, β, $B_{\max}$.

4.6 Experiments on Real-World Data

In this section, we empirically evaluate the backbone method on real-world datasets and compare its performance with various baselines and state-of-the-art alternatives. The metrics and algorithms/software that we use are the same as those outlined in Sections 4.5.2 and 4.5.3, respectively.

4.6.1 Datasets

We experiment on 2 regression and 2 classification datasets from the UCI machine learning repository [92]:

- *Communities and Crime:* This is a regression problem; the goal is to predict the crime rate in various communities in the US [191]. We remove all features whose values are missing in more than 10 data points; then, we remove all data points that still have any missing value. The resulting dataset consists of $n = 1,993$ data points and $p = 100$ features. The original dataset is available at `https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime`.

- *Housing:* This is a regression problem; the goal is to predict housing prices in Boston. The original dataset has no missing values; we expand the dataset by adding squared for all features (except for the binary ones) and interaction terms for all pairs of features. The resulting dataset consists of $n = 506$ data points and $p = 103$ features. The original dataset is available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html#housing`.

- *Breast Cancer:* This is a binary classification problem; the goal is to predict whether a breast cancer instance is benign or malignant [221, 235]. We perform no preprocessing to the dataset, which consists of $n = 599$ data points and $p = 9$ features. The original dataset is available at `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29`.

- *Ionoshphere:* This is a binary classification problem; the goal is to classify radar returns from the ionosphere [200]. We perform no preprocessing to the dataset,

which consists of $n = 351$ data points and $p = 33$ features. The original dataset is available at `https://archive.ics.uci.edu/ml/datasets/Ionosphere`.

In the lines of [132], we append to the data matrix 1,000 random permutations of each (original) feature (column) for both regression and classification instances. By doing so, we have a way to assess the support recovery performance of each method by measuring the fraction of features in the solution that are original features, that is, they were not generated according to the process described above. Moreover, the expanded datasets become truly ultra-high dimensional, which is the regime that we are interested in.

We conduct our experiments as follows. We randomly split, 5 times independently, each original dataset (which we have not expanded yet) into training and testing, at a ratio of $\frac{n_{\text{train}}}{n_{\text{test}}} = 4$. For each split, we expand, 5 times independently, the resulting training and testing sets, using the process outlined above. Therefore, for each split, we obtain 5 different realizations of the noisy features. In total, we conduct 25 experiments per problem. We report both the mean and the standard deviation of each metric.

4.6.2 Sparse Linear Regression

In this section, we present the results for the regression datasets, on which we apply sparse linear regression methods.

We tune `BB` as follows. We select the `screen` function's parameter α such that all but $10,000$ features are eliminated. We set $\beta = 0.5$ and solve $M = 10$ subproblems. We set $B_{\max} = 500$. We solve the subproblems using `SR-REL`; in the m-th subproblem, we cross-validate 3 values for $k_m \in \{10, 20, 30\}$ and set $\gamma_m = \frac{1}{\sqrt{n_m}}$. We solve the reduced problem using `SR` with a time limit of 5 minutes; we cross-validate 5 values for $k \in \{10, 20, 30, 40, 50\}$ and 5 values for the hyperparameter γ.

We benchmark `BB` against some of the methods outlined in Section 4.5.3, tuned as explained below. For `SIS-ENET`, we cross-validate the number of features that are eliminated; then, we apply `ENET` to the selected features, tuned as described in Section 4.3.1; more specifically, we cross-validate 5 values for the hyperparameter $\mu \in [0, 1]$ (the

pure lasso model and the ridge regression model are included in the cross-validation procedure); we discard from the final model any feature whose corresponding regressor has magnitude $\leq 10^{-6}$. For `RFE`, we follow the process outlined in Section 4.5.3 to eliminate features; then, we apply `SR` on the selected features, tuned in the exact same way that we tune `SR` for the reduced problem in `BB`. For `DECO`, we follow the process outlined in Section 4.5.3; we tune `ENET` applied to the selected features exactly as in `SIS-ENET`. Finally, we also compare against an oracle model, whereby we apply `SR` on the original features (i.e., we exclude all noisy features from the model); therefore, this approach can serve as an upper bound for the performance of the remaining methods.

In Table 4.1, we present the results for the communities case study, whereas Table 4.2 reports the results for the housing case study. We make the following observations:

- Out-of-sample R^2: In both case studies, `BB` achieves an increased out-of-sample R^2 compared to `RFE` and `DECO`. In the communities case study, `BB` notably outperforms `SIS-ENET` and approaches the performance of the oracle model. In the housing case study, `SIS-ENET` is particularly effective, in that the screening step manages to eliminate all noisy features, and hence almost matches the performance of `SR-ORACLE`.

- Support recovery accuracy and sparsity: In both case studies, `BB` uses a fraction of 15-20% of original features and results in models with sparsity close to that of `SR-ORACLE`. In terms of the fraction of features used that are noisy, `BB` achieves the lowest rate in the communities case study, but is outperformed by `SIS-ENET` and `RFE` in the housing case study.

- Optimality gap: In both case studies, `BB` achieves an optimality gap that is comparable to that of `SR-ORACLE` and hence returns a solution with optimality guarantees, albeit for the reduced problem.

- Computational time: In both case studies, the solution obtained via `BB` is computed in less than 2 hours, and in time comparable to that of `SR-ORACLE`. `RFE` is substantially slower, taking several hours to run, whereas `SIS-ENET` and `DECO`, which do not involve applying `SR`, are computed in less than 2 minutes.

	R^2	SR-ACC	SR-FA	Sparsity	OG	time (sec)
SR-ORACLE	0.649 (0.017)	32.4 (13.626)	0.0 (0.0)	32.4 (13.626)	0.34 (0.375)	3565.03 (387.337)
SIS-ENET	0.556 (0.035)	43.84 (10.907)	76.773 (8.591)	299.28 (350.675)	-	31.061 (9.529)
RFE	0.57 (0.018)	11.4 (2.082)	71.474 (5.188)	39.96 (0.2)	-	54388.664 (11261.57)
DECO	0.573 (0.031)	29.12 (3.528)	67.178 (12.332)	103.32 (46.992)	-	108.263 (29.504)
BB	0.615 (0.024)	18.36 (4.734)	53.647 (11.718)	40.36 (7.353)	0.147 (0.305)	2505.225 (910.012)

Table 4.1: Results for the communities case study.

	R^2	SR-ACC	SR-FA	Sparsity	OG	T
SR-ORACLE	0.864 (0.05)	41.553 (10.305)	0.0 (0.0)	42.8 (10.614)	0.475 (0.425)	3928.42 (272.247)
SIS-ENET	0.863 (0.038)	50.563 (11.062)	0.0 (0.0)	52.08 (11.394)	-	6.144 (2.066)
RFE	0.712 (0.064)	15.961 (3.761)	58.453 (10.988)	40.4 (4.546)	-	13462.06 (2958.84)
DECO	0.757 (0.055)	27.379 (2.831)	73.791 (14.516)	140.48 (71.93)	-	28.402 (8.599)
BB	0.765 (0.046)	16.66 (3.715)	64.464 (9.278)	48.76 (3.431)	0.531 (0.445)	4597.401 (1318.081)

Table 4.2: Results for the housing case study.

4.6.3 Classification Trees

In this section, we present the results for the classification datasets, on which we apply classification tree methods.

We tune BB as follows. We select the `screen` function's parameter α such that all but $1,000$ features are eliminated. We set $\beta = 0.5$ and solve $M = 15$ subproblems. We set $B_{\max} = 100$. We solve the subproblems using CART; in the m-th subproblem, we cross-validate $D_m \in \{2, 3, 4, 5\}$ (we also cross-validate the minbucket and complexity parameter of CART). We solve the reduced problem using OCT; we cross-validate $D \in \{3, 4, 5, 6, 7, 8\}$ (we also cross-validate the minbucket and complexity parameter of OCT).

We benchmark BB against some of the methods outlined in Section 4.5.3, tuned as explained below. The first two baselines are CART, tuned in the exact same way as OCT in solving the reduced problem for BB, and RF, in which we use 100 trees, we impose no depth limit for the trees (which is common in practice), and we cross-validate the number of features that are considered in each split in each tree between 5 values. We also benchmark against RFE, which is tuned as explained in Section 4.5.3. Finally, we also compare against an oracle model, whereby we apply OCT to the original features (i.e., we exclude all noisy features from the model); therefore, this approach can serve as an upper bound for the performance of the remaining methods.

In Table 4.3, we present the results for the breast cancer case study, whereas Table 4.4 reports the results for the ionosphere case study. We make the following observations:

- Out-of-sample AUC: In the breast cancer case study, BB outperforms CART and RFE, matches the performance of OCT-ORACLE, and achieves a 0.03 lower AUC compared to RF, while still outputting a single, interpretable tree. In the ionosphere case study, the feature selection based methods, namely, BB and RFE, outperform OCT-ORACLE and CART, and compete with RF, having a 0.02 lower AUC.

- Support recovery accuracy and sparsity: In both case studies, BB uses more original features compared to CART and RFE, and results in models that are sparser than those obtained via OCT-ORACLE. In terms of the fraction of features used that are noisy, BB outperforms CART and is outperformed by RFE, which generally results in very shallow trees that use the smallest number of features.

- Computational time: In both case studies, BB's computational time is comparable with that of CART and OCT-ORACLE. Compared to RF and RFE, BB is approximately 5 times faster in the breast cancer case study and more than 30 times faster in the ionosphere case study.

	AUC	**SR-ACC**	**SR-FA**	**Sparsity**	**T**
OCT-ORACLE	0.943 (0.023)	73.778 (11.055)	0.0 (0.0)	6.64 (0.995)	1.364 (0.091)
CART	0.917 (0.031)	28.444 (5.629)	55.353 (18.613)	6.4 (2.102)	6.167 (0.478)
RF	0.972 (0.014)	-	-	-	68.593 (6.154)
RFE	0.933 (0.014)	28.444 (5.629)	20.711 (26.468)	3.84 (2.055)	42.121 (5.391)
BB	0.941 (0.022)	34.222 (9.58)	46.252 (13.927)	5.92 (1.498)	9.815 (0.489)

Table 4.3: Results for the breast cancer case study.

4.7 Conclusions

In this chapter, we developed the backbone method, a novel framework that can be used to train a variety of sparse machine learning models. As we showed, the

	AUC	SR-ACC	SR-FA	Sparsity	T
OCT-ORACLE	0.864 (0.033)	25.939 (6.724)	0.0 (0.0)	8.56 (2.219)	2.639 (0.161)
CART	0.844 (0.021)	6.061 (0.0)	61.825 (11.906)	5.8 (1.893)	31.731 (3.009)
RF	0.891 (0.023)	-	-	-	314.683 (42.199)
RFE	0.877 (0.017)	6.303 (0.839)	11.2 (20.478)	2.56 (1.044)	801.467 (152.225)
BB	0.871 (0.042)	12.606 (1.134)	14.667 (10.887)	4.92 (0.493)	11.399 (0.25)

Table 4.4: Results for the ionosphere case study.

backbone method can accurately and effectively sparsify the set of possible solutions and, as a result, the MIO formulation that exactly models the learning problem can be solved fast for ultra-high dimensional problems. We gave concrete examples of problems where the backbone method can be applied and discussed in detail the implementation details for the sparse regression problem and the decision tree problem. For the sparse regression problem, we showed that, under certain assumptions and with high probability, the backbone set consists of the true relevant features.

As far as the sparse regression problem is concerned, our computational study illustrated that the backbone method outperforms or competes with state-of-the-art methods for ultra-high dimensional problems, accurately scales to problems with $p \sim 10^7$ features in minutes and $p \sim 10^8$ features in hours, and drastically reduces the problem size in problems with $p \sim 10^5$ features hence making the work of exact methods much easier. Regarding the decision tree problem, the backbone method scales to problems with $p \sim 10^5$ features in minutes and, assuming that the underlying problem is indeed sparse (in that only few features are involved in splits in the decision tree), outperforms CART, and competes with random forest, while still outputting a single, interpretable tree. In problems with $p \sim 10^3$, the backbone method can accurately filter the feature set and compute decision trees that match those obtained by applying the state-of-the-art optimal trees framework to the entire problem.

Finally, as we discussed throughout the chapter, the backbone method is generic and can be directly applied to any sparse supervised learning model; examples include sparse support vector machines and sparse principal component analysis. In addition, our proposed framework can be extended to non-supervised sparse machine learning problems, such as the clustering problem. Furthermore, the backbone construction

phase can naturally be implemented in a parallel/distributed fashion.

4.8 Technical Proofs

Proof of Theorem 3. We first restate the model and our main assumptions in Section 4.8. Then, in Section 4.8, we present our high-level approach for proving Theorem 3. Finally, we prove the desired statement in three main steps.

Model & Assumptions

We consider the following model:

- Random design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ such that each row $\boldsymbol{x}_i, i \in [n]$, is an iid copy of the random vector $X = (X_1, ..., X_p) \sim \mathcal{N}(0, \boldsymbol{I}_p)$.
- Fixed but unknown regressors $\boldsymbol{\beta} \in \{-1, 0, 1\}^p$ that satisfy $\|\boldsymbol{\beta}\|_0 \leq k$. Let $\mathcal{S}^{\text{true}}$ denote the set of indices that correspond to the support of the true regressor $\boldsymbol{\beta}$.
- Response vector $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where the noise term $\boldsymbol{\varepsilon}$ consists of iid entries $\varepsilon_i \sim \mathcal{N}(0, \sigma^2), i \in [n]$. Thus, each entry in $\boldsymbol{Y}$ is distributed as $Y = X^\top \boldsymbol{\beta} + \varepsilon \sim \mathcal{N}(0, k + \sigma^2)$ (further assuming $\|\boldsymbol{\beta}\|_0 = k$).

We further assume that $p > n \geq k$ and $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$. When we make asymptotic arguments, we take $p \to \infty$. Given $\mathcal{S} \subseteq [p]$, we denote by $X_\mathcal{S}$ the random vector constructed by selecting from X the entries that are in $\mathcal{S}$. We let $\mathcal{S}_k^p = \{\boldsymbol{s} \in \{0, 1\}^p : \sum_{j \in [p]} s_j \leq k\}$. We analyze a simplified version of the backbone method, which we outline below:

1. Screening step: select top αp features based on their empirical marginal correlation with the response $s_j = \boldsymbol{X}_j^\top \boldsymbol{y}$. Parameter α satisfies $0 < \alpha \leq 1$.
2. Construct M subproblems. In each subproblem, sample uniformly at random $\beta\alpha p$ features among those that survived in Step 1. Parameter β satisfies $0 < \beta \leq 1$. Note that, within each subproblem, features are sampled without replacement, i.e., each feature can appear at most once in a subproblem's feature set.

3. For $m = 1, \ldots, M$, approximately solve sparse regression via its boolean relaxation on the m-th selected subset of features. Let $\mathcal{S}^m$ be the solution to the m-th subproblem.

4. Define the backbone set $\mathcal{B} = \cup_{m=1}^{M} S^m$.

Our goal is to show that, with high probability, $\mathcal{S}^{\text{true}} \subseteq \mathcal{B}$.

High-Level Approach

We consider the following events:

- $\mathcal{E}$: the simplified algorithm fails, namely, $\exists j \in [p]$ such that $j \in \mathcal{S}^{\text{true}}$ and $j \notin \mathcal{B}$.
- $\mathcal{E}_1$: any relevant feature is missed in Step 1.
- $\mathcal{E}_{2,m}$: sparse regression's boolean relaxation fails to recover the relevant features in subproblem $m \in [M]$. Let ε_2 be an upper bound on the probability of this event across all subproblems, i.e., $P(\mathcal{E}_{2,m}) \leq \varepsilon_2, \ \forall m \in [M]$.
- $\mathcal{E}_{3,j}$: relevant feature $j \in \mathcal{S}^{\text{true}}$ is selected in Step 1 but is not selected after Steps 2 and 3.

We fix an arbitrary feature $j \in \mathcal{S}^{\text{true}}$. The number of subproblems in which feature j is sampled follows a binomial distribution with parameters M (number of subproblems) and β (probability of sampling feature j in any subproblem); we denote $Z_j \sim \text{Bino}(M, \beta)$. Then, by the union bound, the probability of failure of the simplified algorithm is at most:

$$\begin{aligned} P(\mathcal{E}) &= P\left[\mathcal{E}_1 \cup \left(\cup_{j \in S^{\text{true}}} \mathcal{E}_{3,j}\right)\right] \\ &\leq P\left(\mathcal{E}_1\right) + \sum_{j \in S^{\text{true}}} P\left(\mathcal{E}_{3,j}\right) \\ &\leq P\left(\mathcal{E}_1\right) + \sum_{j \in S^{\text{true}}} \sum_{m=0}^{M} \varepsilon_2^m P\left(Z_j = m\right) \\ &= P\left(\mathcal{E}_1\right) + k \sum_{m=0}^{M} \varepsilon_2^m P\left(Z = m\right), \end{aligned} \tag{4.10}$$

where $Z \sim \text{Bino}(M, \beta)$.

Analysis of Event $\mathcal{E}_1$

[100] show that, under five conditions and provided that α is sufficiently large, Step 1 succeeds with high probability. The model described in Section 4.8 trivially satisfies four out of these five conditions (e.g., on the eigenvalues of the data covariance matrix, on the magnitude of the nonzero regressors, etc.), whereas the fifth condition requires that $p > n$, $\log p = O(n^{\xi})$ for some $\xi \in (0, 1)$. Then, the following theorem holds:

Theorem 4. *Assume that the data is generated according to the model described in Section 4.8. Then,* $\exists \phi < 1$ *such that, when* $\alpha = O\left(\frac{n^{1-\phi}}{p}\right)$*, we have* $P(\mathcal{E}_1) = O\left[\exp\left(-\frac{n}{\log n}\right)\right]$.

Analysis of Events $\mathcal{E}_{2,m}$

We momentarily turn to the original sparse regression problem. Let $\mathcal{S}$ denote the set of features that sparse regression's boolean relaxation selects when applied to the entire problem (where all p features are considered). [35] prove the following theorem:

Theorem 5. *Let* $\gamma = \frac{1}{n}$*,* $p - k > k$*, and suppose the data are generated as discussed in Section 4.8. Then, for all* $\theta \geq 1$*, for samples* $n \geq \theta(\sigma^2 + 2k)\log(p - k)$*, we have* $\mathbb{P}(\mathcal{S} \neq \mathcal{S}^{true}) = O\left(e^{-\theta}\right)$.

In our setting, we cannot directly apply Theorem 5 since, within each subproblem, we do not necessarily sample all relevant features. We consider an arbitrary subproblem $m \in [M]$ and denote by $\mathcal{P}^m$ the set of features sampled in the m-th subproblem's feature set. When we solve sparse regression's boolean relaxation for the m-th subproblem, we only observe the relevant features in $\mathcal{S}^{\text{true}} \cap \mathcal{P}^m$. Any relevant feature j such that $j \in \mathcal{S}^{\text{true}}$ and $j \notin \mathcal{P}^m$ is viewed as noise in the m-th subproblem. Therefore, we consider the new noise term

$$\varepsilon' = X^{\top}_{\mathcal{S}^{\text{true}} \cap (\mathcal{P}^m)^c} \boldsymbol{\beta}_{\mathcal{S}^{\text{true}} \cap (\mathcal{P}^m)^c} + \varepsilon,$$

which is the sum of at most $k - k_0$ (with $k_0 = |\mathcal{S}^{\text{true}} \cap \mathcal{P}^m|$) independent $\mathcal{N}(0,1)$ random variables and one $\mathcal{N}(0,\sigma^2)$ random variable. Thus, $\varepsilon' \sim \mathcal{N}(0, k - k_0 + \sigma^2)$.

Let $\mathcal{S}^m$ denote the set of features that sparse regression's boolean relaxation selects when applied to subproblem m. Then, by directly applying the result stated in Theorem 5, we obtain the following lemma:

Lemma 11. *Let $\gamma = \frac{1}{n}$, $\beta\alpha p > 2k_0$, and suppose the data are generated as discussed in Section 4.8. Then, for all $\theta \geq 1$, for samples $n \geq \theta(\sigma^2 + 2k)\log(\beta\alpha p) \geq \theta(\sigma^2 + k + k_0)\log(\beta\alpha p - k_0)$, we have*

$$\mathbb{P}(\mathcal{E}_{2,m}) = \mathbb{P}(\mathcal{S}^{true} \cap \mathcal{P}^m \setminus \mathcal{S}^m \neq \emptyset) \leq \mathbb{P}(\mathcal{S}^m \neq \mathcal{S}^{true} \cap \mathcal{P}^m) = O\left(e^{-\theta}\right).$$

Lemma 11 asserts that the probability of not exactly recovering the true support -which is, in fact, more restrictive than our requirement to not miss any relevant feature- decreases exponentially with the parameter θ; as the sample size n increases, we are able to select larger θ and hence obtain tighter guarantees. In the case $k_0 = 0$, we are dealing with a problem with no relevant features, so the upper bound on the probability of error $\mathbb{P}(\mathcal{E}_{2,m})$ is trivially satisfied. Moreover, we note that, since all subproblems are constructed in the same way, the analysis is the same for all $m \in [M]$. This gives the upper bound $\mathbb{P}(\mathcal{E}_{2,m}) \leq \varepsilon_2 = O\left(e^{-\theta}\right)$. Finally, similar to [100], to avoid the selection bias in the screening step, we can split the sample in two halves and use the first for Step 1 and the second for Step 3 of the simplified algorithm.

Analysis of Event $\mathcal{E}_{3,j}$ and Final Result

We now turn back to the last equation in (4.10). By Theorem 4, the first summand converges to 0. We focus on the second summand and apply the binomial theorem to

obtain:

$$\begin{aligned} k\sum_{m=0}^{M}\varepsilon_2^m P\left(Z=m\right) &= k\sum_{m=0}^{M}\varepsilon_2^m\binom{M}{m}\beta^m(1-\beta)^{M-m} \\ &= k(1-\beta)^M\sum_{m=0}^{M}\binom{M}{m}\left(\frac{\varepsilon_2\beta}{1-\beta}\right)^m \\ &= k(1-\beta)^M\left(1+\frac{\varepsilon_2\beta}{1-\beta}\right)^M \\ &= k\left(1-\beta+\varepsilon_2\beta\right)^M. \end{aligned} \tag{4.11}$$

Let us briefly review the quantities that appear in equation (4.11): k is the number of relevant features; ε_2 is an upper bound on the probability of failure for any subproblem and converges to 0 at rate given by Lemma 11; M denotes the number of subproblems that we solve; β denotes the fraction of the αp screened features that we sample in each subproblem.

M and β are parameters of the algorithm; among the two, we believe it is realistic to only control M, since β depends on the available computational resources (e.g., memory). Therefore, we next determine how M has to be selected as function of k, ε_2, β, so as to guarantee that $k\left(1-\beta+\varepsilon_2\beta\right)^M \to 0$. To guarantee a $\frac{1}{\alpha p}$ rate of convergence, we pick

$$k\left(1-\beta+\varepsilon_2\beta\right)^M = O\left(\frac{1}{\alpha p}\right) \tag{4.12}$$

and, therefore,

$$M = O\left(\frac{\log\left(\alpha p k\right)}{\log\left(\frac{1}{1-\beta+\beta\varepsilon_2}\right)}\right). \tag{4.13}$$

This completes the proof of Theorem 3. □

Chapter 5

Velocity and Frequency Estimation in Data Streams

5.1 Introduction

We consider a streaming model of computation [172, 115], where the input is represented as a finite sequence of elements from some finite universe (domain) which is not available for random access, but instead arrives dynamically and one at a time in a stream. We further assume that each element is identified by a unique key and is also associated with a set of features. One of the most fundamental problems in the streaming model is frequency estimation, i.e., given an input stream, estimate the frequency (number of occurrences) of each element. Notice that this can trivially be computed in space equal to the minimum of the universe and the stream size, by simply maintaining a counter for each element or by storing the entire stream, respectively. Nevertheless, data streams are typically characterized by large volume and, therefore, streaming frequency estimation algorithms should require small space, sublinear in both the universe and the stream size. Furthermore, streaming algorithms should generally be able to operate in a single pass (each element should be examined at most once in fixed arrival order) and in real-time (each element's processing time must be low).

Example 2. *Consider a stream of queries arriving on a server. The universe of all elements is the set of all possible queries (of bounded length) and each element is uniquely identified by the query text. Note that any unique query may appear multiple times in the stream. The features associated with a query could include, e.g., the query length, the unigram of the query text (possibly after some pre-processing), etc. The goal is to estimate the frequency distribution of the queries, that is, the number of times each query appears in the stream, in space much smaller than the total number of unique queries.*

Massive data streams appear in a variety of applications. For example, in search query monitoring, Google received more than 1.2 trillion queries in 2012 (which translates to 3.5 billion searches per day) [2]. In network traffic monitoring, AT&T collects over one terabyte of NetFlow [3] measurement data from its production network each day [115]. Moreover, the IPV6 protocol provides nearly 2^{128} addresses, making the universe of possible IP addresses gigantic, especially considering that, in many applications, we are interested in monitoring active IP network connections between pairs (source/destination) of IP addresses. Thus, being able to process a data stream in sublinear space is essential.

Maintaining the frequency distribution of a stream of elements is useful, not only as a sufficient statistic for various empirical measures and functionals (e.g., entropy [56]), but also to identify interesting patterns in the data. An example are the so-called "heavy-hitters" [167, 83], that is, the elements that appear a big number of times, which, e.g., could be indicative of denial of service attacks in network traffic monitoring (see [115] for a detailed discussion of applications).

In this chapter, we address the problem of frequency estimation in data streams, under the additional assumption that a prefix of the input stream has been observed. Along the lines of [137], who address the same problem and extend classical streaming frequency estimation algorithms with a machine learning component, we aim to exploit the observed prefix and the features associated with each element, and develop data-driven streaming algorithms. The proposed algorithms satisfy the small-space requirement, as they significantly compress the input frequency vector, and do operate

in a single pass and in real-time, as their update and query times are constant (except for the training phase, which is more computationally demanding, since we perform optimization and machine learning).

5.1.1 Relevant Literature: Streaming Frequency Estimation Algorithms

A rich body of research has emerged in the streaming model of computation [172, 115]; the first streaming algorithms appeared in the early 1980s, to address, in limited space, problems such as finding the most frequently occurring elements in a stream [167]. A vast literature has since been developed, especially since the 1990s, and numerous problems, including complex machine learning tasks, such as decision tree induction [90], can now be solved in streaming settings.

Sketches [82] are among the most powerful tools to process streaming data. A sketch is a data structure which can be represented as a linear transform of the input. For example, in the context of frequency estimation, the input is the vector of frequencies (or frequency distribution) of the input elements and the sketch is computed by multiplying the frequency distribution by a fixed, "fat" matrix. Of course, for compactness, the matrix that performs the sketch transform is never explicitly materialized and is implicitly implemented via the use of random hash functions.

Any given sketch transform is defined for a particular task. Among the most popular sketching methods for the task of frequency estimation, are the Count-Min Sketch [84] and the Count Sketch [75], which both rely on random hashing and differ in their frequency estimation procedure. Historically, the so-called AMS Sketch [14], which addresses the task of estimating the sum of the squares of the frequencies of the input stream, was among the first sketching algorithms that have been proposed. Sketching algorithms have found numerous applications, including in measuring network traffic [233], in natural language processing [119], in signal processing and compressed sensing [117], and in feature selection [8].

5.1.2 Relevant Literature: Learning-Augmented Streaming Algorithms

The abundance of data that is available today has motivated the development of the field of learning-augmented algorithms, whereby traditional algorithms are modified to leverage useful patterns in their input data. More specifically, in the context of streaming algorithms, [149] and [168] augment with a machine learning oracle the Bloom filter [59, 68], a widely used probabilistic data structure that tests set membership. [137] develop learning-based versions of the Count-Min Sketch and the Count Sketch; a similar approach is taken by [232], who focus on network monitoring and develop a generalized framework to augment sketches with machine learning. The latter two approaches use machine learning in parallel with a standard (random) sketch, that is, they combine a machine learning oracle with standard (conventional) streaming frequency estimation algorithms, such as the Count-Min Sketch.

In this chapter, we consider the same problem as in [137], namely learning-based streaming frequency estimation. However, our approach fundamentally differs in that we combine a (non-random) sketch (i.e., the optimal hashing scheme) and machine learning into a new estimator and hence our approach does not rely on random hashing at all. Instead, we use optimization to learn an optimal (or near-optimal) hashing scheme from (training) data, and machine learning to hash "unseen elements," which did not appear in the training data.

5.1.3 Relevant Literature: Learning to Hash

The proposed approach has connections with the field of learning to hash, a data-dependent hashing approach which aims to learn hash functions from a specific dataset (see [217] for a comprehensive survey). Learning to hash has mostly been considered in the context of nearest neighbor search, i.e., learning a hashing scheme so that the nearest neighbor search result in the hash coding space is as close as possible to the search result in the original space. Optimization-based learning to hash approaches include the works [151, 157, 156]. [17] develop an optimal data-dependent hashing

scheme for the approximate nearest neighbor problem. To the best of our knowledge, our approach is the first that considers learning hashing schemes for the streaming frequency estimation problem, whereby the objective is different.

5.1.4 Relevant Literature: Learning-Augmented Algorithms beyond Streaming and Hashing

Beyond streaming and hashing algorithms, [188] use machine-learned predictions to improve the performance of online algorithms. [160] use reinforcement learning and neural networks to learn workload-specific scheduling algorithms that, e.g., aim to minimize the average job completion time. Machine learning has also been used outside the field of algorithm design, e.g., in signal processing and, specifically, in the context of "structured" (instead of sparse) signal recovery [170] and in optimization. [145] and [20] propose machine learning-based approaches for variable branching in mixed-integer optimization, [144] use reinforcement learning to learn combinatorial optimization algorithms over graphs, [54] use interpretable machine learning methods to learn strategies behind the optimal solutions in continuous and mixed-integer convex optimization problems as a function of their key parameters, and [33] focus specifically on online mixed-integer optimization problems. Machine learning has also been popularized in the context of data management and, in particular, in tasks such as learning index structures [149] and query optimization [150, 180].

5.1.5 Contributions and Outline

Our key contributions can be summarized as follows:

- We develop a novel approach for the problem of frequency estimation in data streams that is based on optimization and machine learning. By exploiting an observed stream prefix, the proposed learning-based streaming frequency estimation algorithm achieves superior performance compared to conventional streaming frequency estimation algorithms.

- We present an exact mixed-integer linear optimization formulation, as well as an efficient block coordinate descent algorithm, that enable us to compute near-optimal hashing schemes and provide a smart alternative to oblivious random hashing schemes. This part of our work could be of independent interest, beyond the problem of frequency estimation in data streams. Further, in a special case, we are able to solve the proposed formulation exactly in linear time using dynamic programming.

- We apply the proposed approach to the problem of search query frequency estimation and evaluate it using both synthetic and real-world data. Computational results indicate that the proposed approach notably outperforms state-of-the-art non-learning and learning-based approaches in terms of its estimation error. Moreover, the proposed approach is by construction interpretable and enables us to get additional insights into the problem of search query frequency estimation.

The rest of the chapter is organized as follows. In Section 5.2, we formalize the streaming frequency estimation problem; we present, at a high level, the Count-Min Sketch, the most widely used random hashing-based streaming frequency estimation algorithm, and the Learned Count-Min Sketch, a learning-augmented version of the Count-Min Sketch; and we give an overview of the proposed approach. In Section 5.3, we formulate the problem of learning the optimal hashing scheme using the observed stream prefix and develop efficient optimization algorithms. Section 5.4 describes the frequency estimation procedure we apply, after the optimal hashing scheme is learned. In Section 5.5, we use synthetic data to explore the performance and scalability of the proposed algorithms and investigate the impact of various design choices on the proposed approach. Section 5.6 empirically evaluates the proposed approach on real-world search query data. Section 5.7 concludes the chapter.

5.2 Preliminaries and Overview of the Proposed Approach

In this section, we formally describe the problem of frequency estimation in data streams and present the state-of-the-art approaches to solving it. Although we explain the notation that we use in the main text, for convenience, we also gather the basic notations in Table 5.1, which includes the basic notations that are used repeatedly throughout the chapter. We note that we generally use the index u to refer to elements, the indices i and k to refer to element IDs, the index j to refer to buckets. For simplicity in notation, elements are referred to using either their symbol u or their ID i, depending on the context; similarly, frequencies are indexed using either of the two approaches.

Formally, we are given input data in the form of an ordered set of elements

$$\mathcal{S} = (u_1, u_2, \ldots, u_{|\mathcal{S}|}),$$

where $u_t \in \mathcal{U},\ \forall t \in [|\mathcal{S}|] := \{1, ..., |\mathcal{S}|\}$, and $\mathcal{U}$ is the universe of input elements. Each element $u \in \mathcal{U}$ is of the form

$$u = (k, \boldsymbol{x}),$$

where (without loss of generality) $k \in [|\mathcal{U}|]$ is a unique ID and $\boldsymbol{x} \in \mathcal{X}$ is a set of features associated with u. The goal is, at the end of $\mathcal{S}$, given an element $u \in \mathcal{U}$, to output an estimate $\tilde{f}_u$ of the frequency

$$f_u = \sum_{t=1}^{|\mathcal{S}|} \mathbb{1}_{(u_t = u)}$$

of that element, i.e., the number of times the element appears in $\mathcal{S}$; here, $\mathbb{1}_{\mathcal{A}}$ denotes the indicator function of event $\mathcal{A}$. We assume that both $\mathcal{S}$ and $\mathcal{U}$ are huge, so we wish to produce accurate estimates in space much smaller than $\min\{|\mathcal{S}|, |\mathcal{U}|\}$. We work under the additional assumption that a prefix $\mathcal{S}_0 = (u_1, u_2, \ldots, u_{|\mathcal{S}_0|})$ (where $|\mathcal{S}_0| \ll |\mathcal{S}|$) of the input stream has already been observed.

Symbol	Explanation
General symbols:	
$\mathcal{U}$	Universe of elements
$\mathcal{U}_0$	Set of elements that appeared in the stream prefix
n	$\lvert\mathcal{U}_0\rvert$
$u \in \mathcal{U}$	Element
$k \in [\lvert\mathcal{U}\rvert]$	Element's unique ID
$\mathcal{X}$	Feature space
$\boldsymbol{x} \in \mathcal{X}$	Element's features
$\mathcal{S} = (u_1, \ldots, u_{\lvert\mathcal{S}\rvert})$	Data stream
$\mathcal{S}_0$	Data stream prefix
f_u	Frequency of element u in $\mathcal{S}$
f_u^0	Frequency of element u in $\mathcal{S}_0$
$\tilde{f}_u$	Estimate of frequency of element u in $\mathcal{S}$
b	Sketch's total buckets
Symbols related to CMS and LCMS:	
w and d	Sketch width and depth
ϕ_j (or ϕ_j^l)	Aggregate frequency in bucket j (or bucket j in level l); this is used in CMS and LCMS
$h_{HH}(\cdot)$	Classifier that decides whether element u is a heavy hitter
Symbols related to the proposed approach:	
$\mathcal{I}_j$	Set of elements in bucket j
c_j	Number of elements in bucket j
μ_j	Mean of frequencies of elements in bucket j
$\boldsymbol{z}_i$	One-hot binary hash code for element with ID i
h_i	Integer hash code for element with ID i
λ	Hyperparameter that controls the trade-off between estimation error and similarity error
$h_S(\cdot)$	Function that maps elements that appeared in the prefix to buckets based on the learned hash code
$h_U(\cdot)$	Classifier that maps elements to buckets

Table 5.1: Notations.

5.2.1 Conventional Approach: Random Sketches

The standard approach to attack this problem is the well-known Count-Min Sketch (CMS) [84], a probabilistic data structure based on random hashing that serves as the frequency table of $\mathcal{S}$. In short, CMS randomly hashes (via a random linear hash function hash$(\cdot)$) each element $u \in \mathcal{U}$ to a bucket in an array $\boldsymbol{\phi}$ of size $w \ll \min\{|\mathcal{S}|, |\mathcal{U}|\}$; whenever element u occurs in $\mathcal{S}$, the corresponding counter $\boldsymbol{\phi}_{hash(u)}$ is

incremented. Since $w \ll |\mathcal{U}|$, multiple elements are mapped to the same bucket and $\boldsymbol{\phi}_{hash(u)}$ overestimates f_u. In practice, multiple arrays $\boldsymbol{\phi}^1, ..., \boldsymbol{\phi}^d$ are maintained (each array is referred to as a "level") and the final estimate for f_u is

$$\tilde{f}_u = \min_{l \in [d]} \boldsymbol{\phi}^l_{hash^l(u)},$$

where $\text{hash}^l(\cdot)$ is the hash function that corresponds to the l-th level. Intuitively, by repeating the estimation procedure multiple times and taking the minimum of the estimated frequencies (all of which overestimate the actual frequency), the resulting estimator's accuracy will improve. CMS provides probabilistic guarantees on the accuracy of its estimates, namely, for each $u \in \mathcal{U}$, with probability $1 - \delta$,

$$|\tilde{f}_u - f_u| \leq \epsilon ||\mathbf{f}||_1,$$

where $\epsilon = \frac{e}{w}$ and $\delta = e^{-d}$. In total, CMS consists of $b = w \times d$ buckets.

5.2.2 Learning-Based Approach: Learned Sketches

To leverage the observed stream prefix, [137] augment the classical CMS algorithm as follows. Noticing that the elements that affect the estimation error the most are the so-called heavy-hitters (i.e., elements that appear many times), they propose to train a classifier

$$h_{\text{HH}} : \mathcal{X} \to \{\text{heavy}, \overline{\text{heavy}}\}$$

that predicts whether an element $u = (k, \boldsymbol{x})$ is going to be a heavy-hitter or not. [1] Then, they allocate b_{heavy} unique buckets to elements identified as heavy-hitters, and randomly allocate the remaining $b_{\text{random}} = b - 2b_{\text{heavy}}$ buckets to the rest of the universe, using, e.g., the standard CMS. We call their algorithm the Learned Count-Min Sketch (LCMS).

[1] [137] identify the heavy-hitters by first predicting the element frequencies (or log-frequencies) using machine learning and then selecting, using validation data, the optimal cutoff threshold for an element to be considered a heavy-hitter. In their experiments, they predict whether an item is in the top 1% of the frequencies.

An important remark is that each of the b_{heavy} unique buckets allocated to heavy-hitters should maintain both the frequency and the ID of the associated element. As explained, this can be achieved by using hashing with open addressing, whereby it suffices to store IDs hashed into $\log b_{\text{heavy}} + t$ bits (instead of whole IDs which could be arbitrarily large) to ensure there is no collision with probability $1 - 2^{-t}$. Noticing that $\log b_{\text{heavy}} + t$ is comparable to the number of bits per counter, the space for a unique bucket is twice the space of a normal bucket. The learning augmented algorithm is shown to outperform, both theoretically and empirically, its conventional, fully-random counterpart. Additionally, they prove that under certain distributional assumptions, allocating unique buckets to heavy-hitters is asymptotically optimal [137, 4]. In general, however, their approach remains heuristic, does not guarantee optimal performance, and possibly throws away information by taking hard, binary decisions.

5.2.3 Overview of the Proposed Approach

Motivated by the success of LCMS, we investigate an alternative, optimization-based approach in using the observed stream prefix to enhance the performance of the frequency estimator.

At a high level, the proposed two-phase approach works as follows. In the first phase, the elements that appeared in the stream prefix are optimally allocated to buckets based on their observed frequencies so that the frequency estimation error is minimized and, at the same time, similar elements are mapped to the same bucket. Importantly, contrary to CMS-based approaches, in the proposed approach, the estimate for an element's frequency is the average of the frequencies of all elements that are mapped to the same bucket. Therefore, we aim to assign "similar" elements to the same bucket. In the second phase, once we have an optimal allocation of the elements that appeared in the prefix to buckets, we train a classifier mapping elements to buckets based on their features. By doing so, we are able to provide estimates for unseen elements that did not appear in the prefix and hence their frequencies are not recorded.

The proposed hashing scheme consists of a hash table mapping IDs of elements that appeared in the prefix to buckets and the learned classifier. In addition, for each bucket, we need to maintain the sum of frequencies of all elements mapped therein. During stream processing, that is, once the estimator is ready, whenever an element that had appeared in the prefix re-appears, we increment the counter (i.e., the aggregated frequency) of the bucket to which the element was mapped. Finally, to answer count-queries for any given element, we simply output the current average frequency of the bucket where the element is mapped (either via the hash table or via the classifier).

Figure 5-2 provides the flowcharts for the proposed approach. In particular, Figure 5-1(a) corresponds to the learning phase, where the stream prefix is used to learn the optimal hashing scheme and the classifier; Figure 5-1(b) illustrates how the proposed approach answers count queries for any input element; Figures 5-2(a) and 5-2(b) show the update mechanism of the proposed approach without and with the use of Bloom filters, respectively.

5.3 Learning the Optimal Hashing Scheme

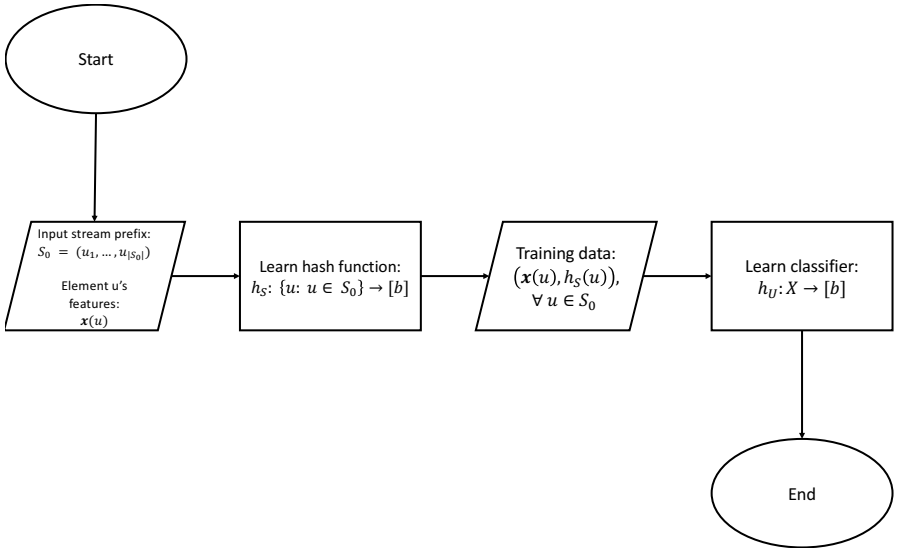
In this section, we develop the proposed approach in learning the optimal hashing scheme.

5.3.1 Exact Formulation

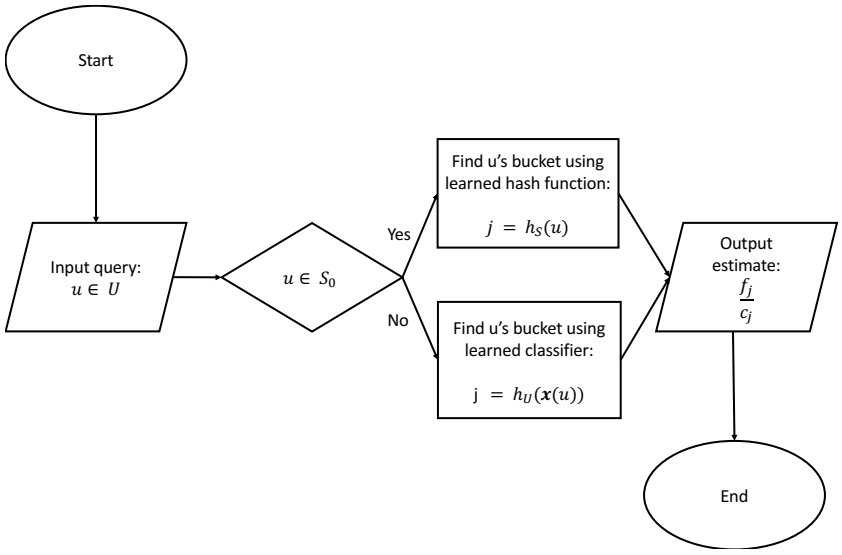
Let $\mathcal{S}_0 = (u_1, ..., u_{|\mathcal{S}_0|})$ be the observed stream prefix. We denote by f_u^0 the empirical frequency of element u in $\mathcal{S}_0$, i.e.,

$$f_u^0 = \sum_{t=1}^{|\mathcal{S}_0|} \mathbf{1}_{(u_t = u)},$$

and by $\boldsymbol{f}^0(\mathcal{S}_0)$ the entire frequency distribution after observing $\mathcal{S}_0$. Moreover, $\mathcal{U}_0 = \{u \in \mathcal{U} : \ f_u^0 > 0\}$ is the set of all distinct elements that appeared in $\mathcal{S}_0$ and let $|\mathcal{U}_0| = n$. We introduce $n \times b$ binary variables, where b is the total number of available

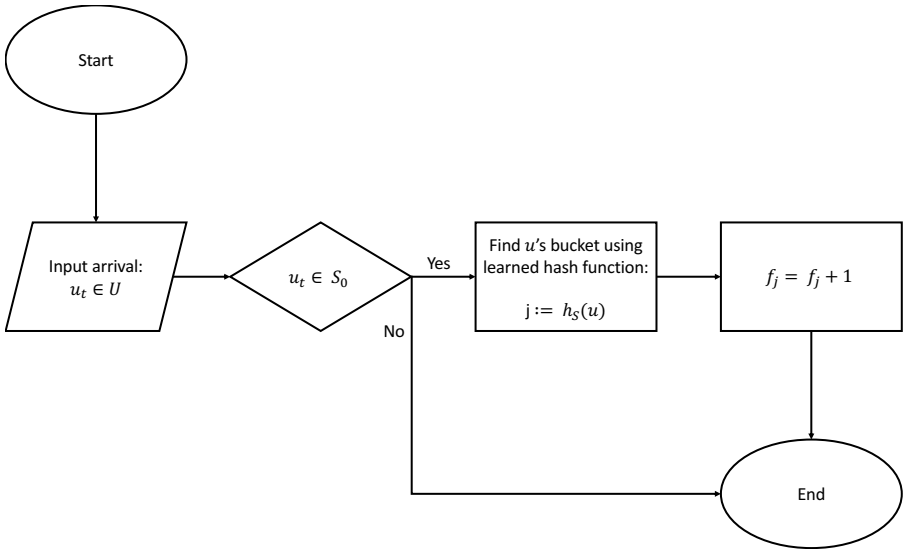


(a) Learning the optimal hashing scheme

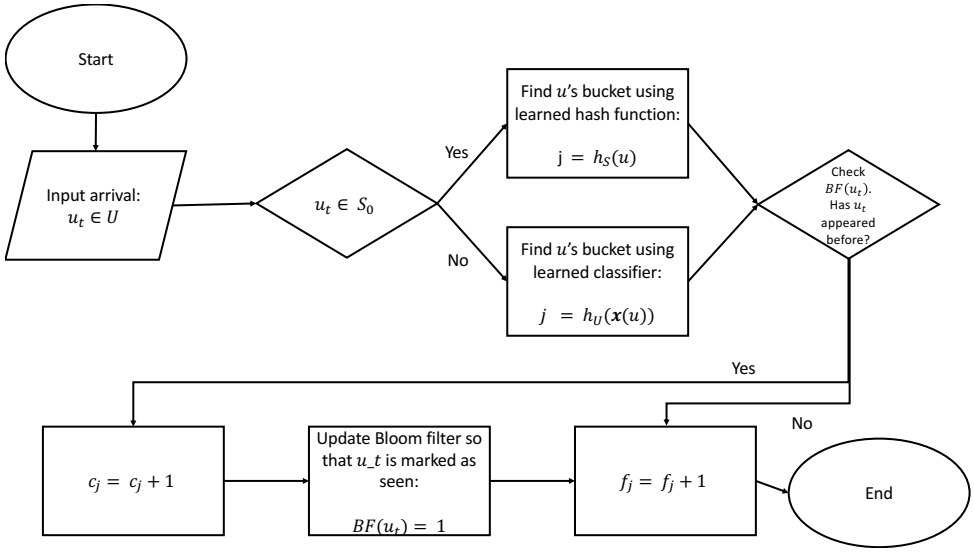


(b) Answering count queries for element $u \in \mathcal{U}$

Figure 5-1: Flowcharts for the proposed approach (I).



(a) Updating sketch at time t upon arrival of element $u_t \in \mathcal{U}$



(b) Updating sketch with Bloom filter extension at time t upon arrival of element $u_t \in \mathcal{U}$

Figure 5-2: Flowcharts for the proposed approach (II).

buckets, defined as

$$z_{ij} = \begin{cases} 1, & \text{if } i\text{th element of } \mathcal{U}_0 \text{ is mapped to bucket } j, \\ 0, & \text{otherwise.} \end{cases}$$

Each row $\boldsymbol{z}_i$ of Z (where we denote $[Z]_{ij} = z_{ij}$) can be viewed as an one-hot binary hash code mapping element i to one of the buckets. At the end of the stream and given a fixed assignment for the variables z_{ij}, the final estimate of the frequency of element $i \in [n]$ is

$$\tilde{f}_i = \sum_{j \in [b]} z_{ij} \left(\frac{\sum_{k \in [n]} z_{kj} f_k}{\sum_{k \in [n]} z_{kj}} \right).$$

The resulting, e.g., absolute estimation error is $\sum_{i \in [n]} |\tilde{f}_i - f_i|$; a natural objective is to pick the variables z_{ij} that minimize this absolute error in the observed stream prefix. An alternative objective we could pick is the expected magnitude of the absolute error $\frac{1}{\sum_{k \in [n]} f_k} \sum_{i \in [n]} f_i \cdot |\tilde{f}_i - f_i|$, whereby it is assumed that the probability p_i of observing element i is equal to its empirical probability in the observed stream prefix, i.e., $p_i := \frac{f_i}{\sum_{k \in [n]} f_k}$. In fact, this metric is used by [137] in their theoretical analysis. However, such an approach would heavily weigh the most frequently occurring elements and would probably produce highly inaccurate estimates for less frequent elements. As we would like to achieve a uniformly small estimation error, we stick to the former objective and select the variables z_{ij} that solve the optimization formulation which we will present shortly. We incorporate an additional term in the objective function of the proposed formulation, to take the features associated with each element into account when computing the optimal mapping of elements to buckets. For $\lambda \in [0, 1]$,

we have:

$$\begin{aligned}\min_{Z\in\{0,1\}^{n\times b}} \quad & \sum_{i\in[n]}\sum_{j\in[b]} z_{ij}\left[\lambda\left|f_i^0 - \frac{\sum_{k\in[n]} z_{kj} f_k^0}{\sum_{k\in[n]} z_{kj}}\right|\right. \\ & \qquad \left. +(1-\lambda)\sum_{k\in[n]} z_{kj}\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2\right] \qquad (5.1)\\ \text{s.t.} \quad & \sum_{j\in[b]} z_{ij} = 1, \quad \forall i \in [n].\end{aligned}$$

The parameter $\lambda \in [0, 1]$ controls the trade-off between hashing schemes that map to the same bucket elements that are similar in terms of their observed frequencies in the prefix ($\lambda \to 1$) and hashing schemes that put more weight on the elements' feature-wise similarity ($\lambda \to 0$). Therefore, we refer to the first term in the objective as the estimation error and to the second term as the similarity error.

Problem (5.1) is a nonlinear binary optimization problem, so it is, in principle, hard to solve. Therefore, we next develop different approaches that can be used to solve it to optimality or near-optimality in different regimes.

5.3.2 Mixed-Integer Linear Reformulation

As we show next, Problem (5.1) can be as reformulated as a mixed integer linear optimization problem by introducing auxiliary variables and new constraints. Formally, we have the following lemma (the proof is presented in Section 5.8):

Lemma 12. *Problem* (5.1) *is equivalent with the following mixed-integer linear optimization problem:*

$$
\begin{aligned}
\min_{\substack{Z\in\{0,1\}^{n\times b},\\ E\in\mathbb{R}^{n\times b}_{\geq 0},\\ \Theta\in\mathbb{R}^{n\times n\times b}_{\geq 0},\\ \Delta\in[0,1]^{n\times n\times b}}} \quad & \sum_{i\in[n]}\sum_{j\in[b]}\left[\lambda\theta_{iij} + (1-\lambda)\sum_{k\in[n]}\delta_{ikj}\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2\right] \\
s.t. \quad & \sum_{j\in[b]} z_{ij} = 1, \\
& \qquad\qquad \forall i \in [n], \\
& \sum_{k\in[n]}\theta_{ikj} - f_i^0\sum_{k\in[n]} z_{kj} + \sum_{k\in[n]} f_k^0 z_{kj} \geq 0, \\
& \qquad\qquad \forall i \in [n],\ \forall j \in [b], \\
& \sum_{k\in[n]}\theta_{ikj} + f_i^0\sum_{k\in[n]} z_{kj} - \sum_{k\in[n]} f_k^0 z_{kj} \geq 0, \\
& \qquad\qquad \forall i \in [n],\ \forall j \in [b], \\
& \theta_{ikj} \geq e_{ij} - M(1 - z_{kj}), \\
& \qquad\qquad \forall i \in [n],\ \forall k \in [n],\ \forall j \in [b], \\
& \theta_{ikj} \leq e_{ij}, \\
& \qquad\qquad \forall i \in [n],\ \forall k \in [n],\ \forall j \in [b], \\
& \theta_{ikj} \leq M z_{kj}, \\
& \qquad\qquad \forall i \in [n],\ \forall k \in [n],\ \forall j \in [b], \\
& \delta_{ikj} \geq z_{ij} + z_{kj} - 1, \\
& \qquad\qquad \forall i \in [n],\ \forall k \in [n],\ \forall j \in [b], \\
& \delta_{ikj} \leq z_{ij}, \\
& \qquad\qquad \forall i \in [n],\ \forall k \in [n],\ \forall j \in [b], \\
& \delta_{ikj} \leq z_{kj}, \\
& \qquad\qquad \forall i \in [n],\ \forall k \in [n],\ \forall j \in [b],
\end{aligned}
\tag{5.2}
$$

where M is a constant that satisfies $M \geq \max_{i\in[n]} f_i^0$.

Problem (5.2) consists of $\mathcal{O}(n^2 b)$ variables and constraints. As our computational

study in Section 5.5 suggests, by solving the reformulated Problem (5.2), we are able to compute optimal hashing schemes for problems with thousands of elements. Nevertheless, solving a mixed integer linear optimization problem of that size can still be prohibitive in the applications we consider. For example, in the real-world case study in Section 5.6, we map up to tens of thousands of elements to up to thousands of buckets, so Formulation (5.2) would consist of variables and constraints in the order of 10^{11}. Therefore, we next develop a tailored block coordinate descent algorithm that works well in practice.

5.3.3 Efficient Block Coordinate Descent Algorithm

By exploiting the problem structure, we propose the following efficient block coordinate descent algorithm (Algorithm 6) that can be used to either heuristically solve Problem (5.1) or compute high-quality warm starts for Problem (5.2).

Concerning the algorithm's initialization, we start from a random allocation of elements to buckets. Alternatively, we could sort elements in $\mathcal{U}_0$ in terms of their observed frequencies and allocate the first $\left\lceil \frac{\mathcal{U}_0}{b} \right\rceil$ elements to the first bucket, the next $\left\lceil \frac{\mathcal{U}_0}{b} \right\rceil$ to the second bucket, and so forth, or we could even use the heavy-hitter heuristic (that is, assign heavy-hitters to their own bucket and the remaining elements at random).

In our implementation, we maintain, for each bucket, the set of elements $\mathcal{I}_j$ mapped therein, its cardinality c_j and mean frequency μ_j, as well as the associated estimation error $e_j = \sum_{i \in \mathcal{I}_j} |f_i^0 - \mu_j|$ and similarity error $s_j = \sum_{(i,k) \in \mathcal{I}_j \times \mathcal{I}_j} \|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2$. After any update performed by Algorithm 6 we only need to update the above quantities, instead of having to recompute them from scratch and, therefore, we can directly evaluate the objective function value ε associated with any particular mapping of elements to buckets.

In each iteration, Algorithm 6 examines sequentially and in random order all n blocks of b variables $\boldsymbol{z}_i,\ i \in [n]$. Notice that each block contains all possible mappings of a particular element to any bucket. For each element i, we greedily select the mapping that minimizes the overall estimation error. To do so, we remove element i

Algorithm 6: Block Coordinate Descent Algorithm.

Input: Observed frequency vector $\boldsymbol{f}^0 \in \mathbb{N}^n$, number of buckets $b \in \mathbb{N}$, hyperparameter $\lambda \in [0,1]$.
Output: Learned one-hot hashing scheme $Z \in \{0,1\}^{n \times b}$.

Initialize Z satisfying $\sum_{j \in [b]} z_{ij} = 1, \forall i \in [n]$
$\varepsilon_0 \leftarrow 0$ ▷ `Objective function value for initial map`
for $j \in [b]$ **do**
  ▷ `Find set of elements, cardinality, and mean for bucket` j `in initial map:`
  $\mathcal{I}_j, c_j, \mu_j \leftarrow \{i \in [n] : z_{ij} = 1\}, |\mathcal{I}_j|, \frac{\sum_{i \in \mathcal{I}_j} f_i^0}{c_j}$
  ▷ `Compute estimation error` e_j `and similarity error` s_j `for bucket` j `in initial map:`
  $e_j, s_j \leftarrow \sum_{i \in \mathcal{I}_j} \left| f_i^0 - \mu_j \right|, \sum_{(i,k) \in \mathcal{I}_j \times \mathcal{I}_j} \|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2$
  $\varepsilon_0 \leftarrow \varepsilon_0 + [\lambda e_j + (1-\lambda) s_j]$
end for
$t \leftarrow 0$
repeat
  Draw a random permutation $\boldsymbol{\sigma}$ of the set $[n]$
  for $i \in [n]$ **do**
    for $j \in [b]$ **do**
      ▷ `Check if` σ_i `is already in bucket` j `and compute error with and without` σ_i`:`
      if $\sigma_i \in \mathcal{I}_j$ **then**
        $\varepsilon_{\sigma_i, j} \leftarrow \lambda e_j + (1-\lambda) s_j$
        $\varepsilon_{-\sigma_i, j} \leftarrow \lambda \left(\sum_{k \in \mathcal{I}_j \setminus \{\sigma_i\}} \left| f_k^0 - \frac{c_j \mu_j - f_{\sigma_i}^0}{c_j - 1} \right| \right) + (1-\lambda) \left(s_j - 2 \sum_{k \in \mathcal{I}_j} \|\boldsymbol{x}_{\sigma_i} - \boldsymbol{x}_k\|^2 \right)$
        ▷ `Update bucket` j `stats and errors after removing` σ_i`:`
        $\mathcal{I}_j, c_j, \mu_j \leftarrow \mathcal{I}_j \setminus \{\sigma_i\}, c_j - 1, \frac{c_j \mu_j - f_{\sigma_i}^0}{c_j - 1}$
        $e_j, s_j \leftarrow \sum_{k \in \mathcal{I}_j} \left| f_k^0 - \mu_j \right|, s_j - 2 \sum_{k \in \mathcal{I}_j} \|\boldsymbol{x}_{\sigma_i} - \boldsymbol{x}_k\|^2$
      else
        $\varepsilon_{\sigma_i, j} \leftarrow \lambda \left(\sum_{k \in \mathcal{I}_j \cup \{\sigma_i\}} \left| f_k^0 - \frac{c_j \mu_j + f_{\sigma_i}^0}{c_j + 1} \right| \right) + (1-\lambda) \left(s_j + 2 \sum_{k \in \mathcal{I}_j} \|\boldsymbol{x}_{\sigma_i} - \boldsymbol{x}_k\|^2 \right)$
        $\varepsilon_{-\sigma_i, j} \leftarrow \lambda e_j + (1-\lambda) s_j$
      end if
    end for
    ▷ `Find best bucket` $j^\star$ `and update stats and errors after mapping` σ_i `to it:`
    $j^\star \leftarrow \operatorname{argmin}_{j \in [b]} \varepsilon_{\sigma_i, j} + \sum_{\ell \in [b] \setminus \{j\}} \varepsilon_{-\sigma_i, \ell}$
    $\boldsymbol{z}_i \leftarrow \boldsymbol{e}_{j^\star}$ ▷ $\boldsymbol{e}_{j^\star}$ `denotes the` $j^\star$`-th standard unit vector`
    $\mathcal{I}_{j^\star}, c_{j^\star}, \mu_{j^\star} \leftarrow \mathcal{I}_{j^\star} \cup \{\sigma_i\}, c_{j^\star} + 1, \frac{c_{j^\star} \mu_{j^\star} + f_{\sigma_i}^0}{c_{j^\star} + 1}$
    $e_{j^\star}, s_{j^\star} \leftarrow \sum_{k \in \mathcal{I}_{j^\star}} \left| f_k^0 - \mu_{j^\star} \right|, s_{j^\star} + \sum_{k \in \mathcal{I}_{j^\star}} \|\boldsymbol{x}_{\sigma_i} - \boldsymbol{x}_k\|^2$
  end for
  $t \leftarrow t + 1$
  $\varepsilon_t \leftarrow \sum_{j \in [b]} [\lambda e_j + (1-\lambda) s_j]$
until $\varepsilon_{t-1} - \varepsilon_t < \epsilon$
return Z

from its current bucket and compute the estimation error associated with each bucket j, first with element i allocated to bucket j and then without element i. We allocate element i to the bucket $j^\star$ that minimizes the sum of all error terms.

The algorithm terminates when the improvement in estimation error is negligible; in case we are willing to obtain an intermediate solution faster, the termination criterion can be set to a user-specified maximum number of iterations. As we empirically show, Algorithm 6 converges to a local optimum after a few tens of iterations and produces high-quality solutions. Given that algorithm is not guaranteed to converge to a globally optimum solution, the process can be repeated multiple times from different starting points.

Algorithm 6 can be efficiently implemented so that the complexity of each iteration is $\mathcal{O}(n^2 b)$. This is to be expected since, for each bucket, we need to compute the similarity error between all pairs of elements mapped therein, which requires $\mathcal{O}(n^2 b)$ operations.

5.3.4 The $\lambda = 1$ Case: Efficient Dynamic Programming Algorithm

In the special case where we set $\lambda = 1$, that is, we do not take the features into account when computing the optimal hashing scheme, we obtain the following formulation:

$$\begin{aligned} \min_{Z \in \{0,1\}^{n \times b}} \quad & \sum_{i \in [n]} \sum_{j \in [b]} z_{ij} \left| f_i^0 - \frac{\sum_{k \in [n]} z_{kj} f_k^0}{\sum_{k \in [n]} z_{kj}} \right| \\ \text{s.t.} \quad & \sum_{j \in [b]} z_{ij} = 1, \quad \forall i \in [n]. \end{aligned} \tag{5.3}$$

Problem (5.3) is an one-dimensional k-median clustering problem and has been thoroughly studied in the literature. It is fairly straightforward to develop an $O(n^2 b)$ dynamic programming algorithm to solve Problem (5.3) to provable optimality as per [216]. An even more efficient solution method for Problem (5.3) has been developed in the context of optimal quantization; using dynamic programming in combination

with a matrix searching technique, [223] solves Problem (5.3) to optimality in $O(nb)$ time. We refer the interested reader to [121] for a detailed and unified presentation of the above methods.

Given that we can obtain an optimal solution to Problem (5.3) very fast, in $O(nb)$ time, we propose to use it as a warm start for the general $\lambda \in [0, 1)$ case. Therefore, we provide another alternative for the initialization step of Algorithm 6, in addition to the ones discussed in Section 5.3.3.

5.4 Frequency Estimation

In this section, we describe the frequency estimation component of the proposed estimator, which, in its simplest form, consists of a multi-class classifier.

5.4.1 Frequency Estimation for Elements Seen in the Prefix

Once the optimal assignment Z is computed, we essentially have a hash code $h_i = \sum_{j \in [b]} j \cdot \mathbb{1}_{(z_{ij}=1)},\ i \in [n]$, for each element $u \in \mathcal{U}_0$. Therefore, for element $u \in \mathcal{U}_0$, indexed by $i \in [n]$, we simply estimate its frequency as

$$\tilde{f}_i = \frac{\sum_{k \in [n]: h_k = h_i} f_k}{\sum_{k \in [n]: h_k = h_i} 1} = \mu_j.$$

We denote by $h_{\mathrm{S}} : \mathcal{U}_0 \to [b]$ the function that maps elements seen in the prefix to buckets according to the learned hash code.

5.4.2 Similarity-Based Frequency Estimation for Unseen Elements

To be able to produce frequency estimates for elements that did not appear in the prefix, i.e., $u \in \mathcal{U} \setminus \mathcal{U}_0$, we formulate a multi-class classification problem, mapping

elements to buckets based on their features. Formally, we search for a function

$$h_{\mathrm{U}} : \mathcal{X} \to [b].$$

The training set consists of all data points in

$$\{(\boldsymbol{x}_i, h_i) : u_i = (k_i, \boldsymbol{x}_i) \in \mathcal{U}_0\},$$

that is, all feature-hash code tuples for elements that appeared in the prefix. Such a classifier will allow us to estimate the frequencies of unseen elements based on the average of the frequencies of elements that "look" similar. The estimate for element $u = (k, \boldsymbol{x}) \in \mathcal{U} \setminus \mathcal{U}_0$ is then

$$\tilde{f}_u = \frac{\sum\limits_{k \in [n] : h_k = h_{\mathrm{U}}(x)} f_k}{\sum\limits_{k \in [n] : h_k = h_{\mathrm{U}}(x)} 1}.$$

5.4.3 Adaptive Counting Extension: Keeping Track of the Frequencies of Unseen Elements

So far, we have described a static approach; we learn the optimal hashing scheme for the elements that appear in the stream prefix and then keep track only of their frequencies. The estimated frequencies for all elements are based only on the frequencies of elements in $\mathcal{U}_0$ (which appeared in S_0). We next describe a dynamic approach, that keeps track of the frequencies of elements beyond the ones in $\mathcal{U}_0$. At a high level, the adaptive approach is based on approximately counting the distinct elements in each bucket. We work as follows.

1. We learn the optimal hashing scheme based on the observed stream prefix and train a classifier mapping elements to buckets, as outlined above. For each bucket, we only record the number of elements that are mapped therein (instead of storing the IDs of the elements that are mapped to this bucket). We use the classifier to determine which bucket any element is mapped to.

2. We maintain a Bloom filter [59] BF, i.e., a probabilistic data structure that,

given a universe of elements $\mathcal{U}$ and a set $\mathcal{U}' \subseteq \mathcal{U}$, probabilistically tests, for any element $u \in \mathcal{U}$, whether $u \in \mathcal{U}'$ (here, $\mathcal{U}'$ corresponds to the elements that have appeared in the stream). If $u \in \mathcal{U}'$, then we deterministically have that $\text{BF}(u) = 1$. However, if $u \notin \mathcal{U}'$, then it need not be the case that $\text{BF}(u) = 0$ (therefore a Bloom filter is prone to false positives - we will explain the impact of those in the sequel).

3. We initialize the Bloom filter based on the elements $u \in \mathcal{U}_0$. Therefore, all elements $u \in \mathcal{U}_0$ will initially have $\text{BF}(u) = 1$. On the other hand, elements $u \notin \mathcal{U}_0$ may initially have either $\text{BF}(u) = 0$ or $\text{BF}(u) = 1$.

4. For every subsequent element u that appears in the stream after the stream prefix S_0 has been processed, we map it to a bucket $j \in [b]$ using the trained classifier. Then, we test whether we have already seen u, using the Bloom filter. If $\text{BF}(u) = 0$, we increase both the frequency ϕ_j and the number of elements c_j in the bucket j, and we set $\text{BF}(u) = 1$. If $\text{BF}(u) = 1$, we only increase the frequency ϕ_j.

5. When queried for the frequency of any element $u \in \mathcal{U}$, regardless of whether it appeared in $\mathcal{U}_0$ or not, we estimate

$$\tilde{f}_u = \frac{\phi_j}{c_j} BF(u),$$

where j is the bucket in which u is mapped using the classifier.

The impact of Bloom filters' false positives is that the proposed approach will mark as seen elements that have not appeared in the stream. When one such element actually appears in the stream, we will not increase the counter c_j that tracks the number of elements in the bucket j where this element is mapped. Therefore, the estimated number of elements c_j in bucket j will be less than the actual number. As a result, the adaptive counting extension will generally overestimate elements' frequencies.

The flowchart for the adaptive counting extension of the proposed approach is given in Figure 5-2(b) in Section 5.2.3.

5.5 Experiments on Synthetic Data

In this section, we empirically evaluate the proposed approach on synthetic data. We investigate the performance and scalability of the optimization approaches discussed in Section 5.3, and explore the possibility of using different classifiers for unseen elements (as per Section 5.4).

5.5.1 Data Generation Methodology

The data that we use in our synthetic experiments are generated according to the following methodology:

- Elements: We parameterize the universe of elements $\mathcal{U}$ by a positive integer $G \in \mathbb{Z}_{>0}$ that controls the problem size in the way that we explain next. We generate G groups of elements $\mathcal{G}_1, \ldots, \mathcal{G}_G$ of exponentially increasing sizes $2^{G_0+1}, \ldots, 2^{G_0+G}$ (where $G_0 \in \mathbb{Z}_{\geq 0}$ is an additional parameter that determines the size of the smallest group; we use $G_0 = 2$ in our experiments). We associate each group $\mathcal{G}_g, g \in [G]$, with a p-dimensional normal distribution (we use $p = 2$ in our experiments to enable visualization) with mean $\boldsymbol{\mu}_g$ selected uniformly at random from $[-10, 10]^p$ and covariance matrix equal to the identity. We draw the features associated with each element $u \in \mathcal{G}_g$ as a realization of the p-dimensional normal distribution $\mathcal{N}(\boldsymbol{\mu}_g, I)$ that corresponds to the element's group.

- Stream: We generate the data stream $\mathcal{S}$ according to the following process. We associate each group $\mathcal{G}_g, g \in [G]$ with an arrival probability that is proportional to $\frac{1}{g}$. Within group $\mathcal{G}_g$, we assign to each element $u \in \mathcal{G}_g$ a uniform probability of arrival $\frac{1}{|\mathcal{G}_g|}$. Thus, smaller groups are more likely to appear and elements therein have a larger probability of selection so that they represent the heavy

hitters. We construct the stream by first selecting the group that each new arrival belongs to and then selecting the actual element from within that group. As far as the stream prefix $\mathcal{S}_0$ is concerned, we would want to mimic a real-world scenario where not all elements from within each group start appearing since the beginning of the stream. Therefore, when we generate the prefix, we only allow for a fraction $g_0 \in [0, 1]$ of elements to be selected from within each group $\mathcal{G}_g, g \in [G]$, each with probability $\frac{1}{g_0|\mathcal{G}_g|}$. Finally, we remark that, in our experiment, we generate a stream prefix of size $|\mathcal{S}_0| = 10 \cdot 2^G$.

For example, by setting $G = 10$ and $g_0 = 0.5$, we obtain a problem with $8,192$ elements, out of which we only allow for $4,096$ to appear in the prefix, which in turn has size $10,240$. Therefore, we aim to learn a hashing scheme that maps at most $4,096$ elements to 10 buckets; the memory requirements of such a hashing scheme would be ≈ 20 KB.

5.5.2 Algorithms and Software

We next summarize the algorithms and software that we use in our experiments. We note that all algorithms were implemented in Python 3 and all experiments were performed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running CentOS release 7. We independently repeat each experiment 10 times and report the averaged error, as well as its standard deviation.

We implement and refer to the optimization algorithms presented in Section 5.3 as follows:

- `milp`: Solves the mixed-integer linear optimization problem (Problem (5.2)) from Section 5.3.2 using the commercial MIO solver Gurobi [123].

- `bcd`: Implements the block coordinate descent algorithm (Algorithm 6) from Section 5.3.3.

- `dp`: Solves Problem (5.3) in linear time via dynamic programming (Section 5.3.4) using a Python wrapper for the R package Ckmeans.1d.dp [216].

The machine learning algorithms that we examine include a linear classifier, namely, multinomial logistic regression (`logreg`), a tree-based classifier, namely, CART (`cart`) [63], and an ensemble classifier, namely, random forest (`rf`) [62]. All methods are tuned using 10-fold cross validation; the hyperparameters that we tune are the weight of a ridge regularization term for `logreg`, the minimum impurity decrease and the maximum depth for `cart`, the maximum number of features in each split and the maximum depth for `rf`. Unless stated otherwise, we use `cart` as the underlying classifier in our experiments. We use the Scikit-learn machine learning package's implementation of all the above algorithms [183].

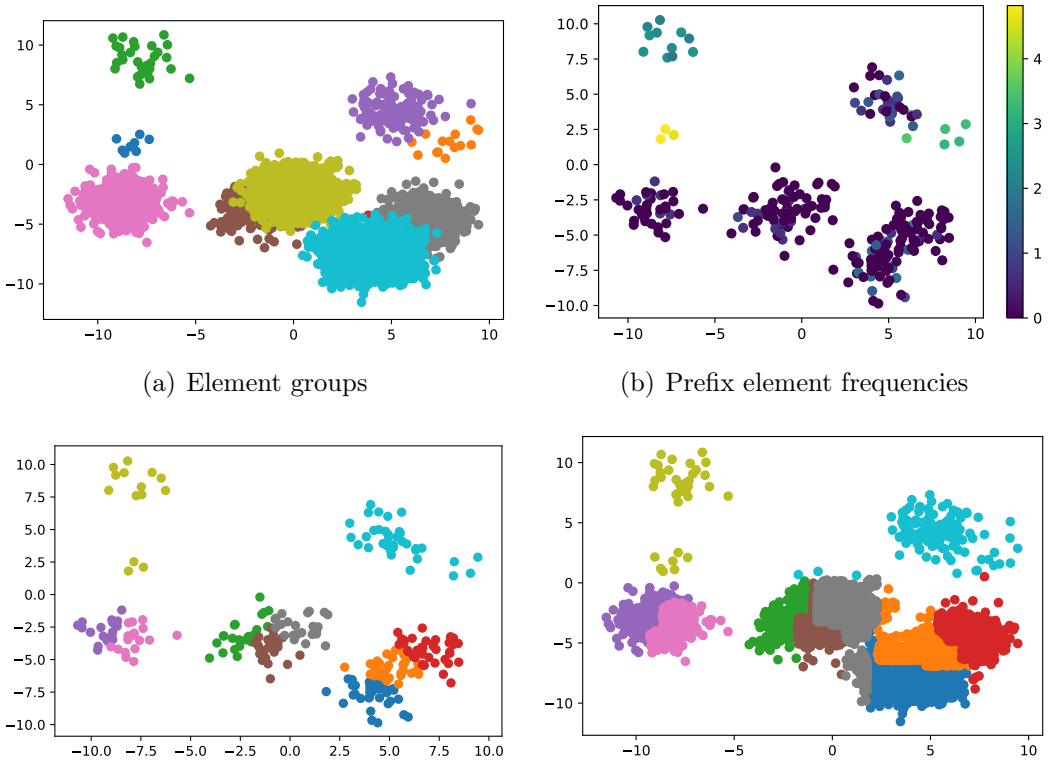
Finally, we use the following notation for the frequency estimation algorithms presented in this chapter. We refer to the proposed estimator as `opt-hash`. We refer to CMS (the standard Count-Min Sketch) as `count-min` and to LCMS (the learned Count-Min Sketch with the heavy-hitter heuristic) as `heavy-hitter`. We implement all the above estimators in Python.

5.5.3 Visualization: Learned Hash Code for Seen and Unseen Elements

In Figure 5-3, we show an instance of a synthetically generated problem with $G = 10$ groups (Figure 5-3(a) colors elements depending on their actual group). Figure 5-3(b) shows the logarithm of the frequency of each element that appeared in a prefix of length $|\mathcal{S}_0| = 1,000$; we assume that a fraction of $g_0 = 0.33$ elements from each group can appear in the prefix. In Figure 5-3(c), we present the learned hash code for elements that actually appeared in the prefix (using the `bcd` algorithm), whereas Figure 5-3(d) illustrates the hash code predicted for unseen elements (using `cart`).

5.5.4 Results

We next present the results from our computational study on synthetic data. Let Z denote the learned hash code; for elements $i \in \mathcal{S}_0$, $\boldsymbol{z}_i$ is obtained using one of the algorithms presented in Section 5.3; for elements $i \notin \mathcal{S}_0$, $\boldsymbol{z}_i$ is obtained



(a) Element groups

(b) Prefix element frequencies

(c) Hash code for elements that appeared in the prefix

(d) Hash code for unseen elements

Figure 5-3: Visualization of element groups and hash codes.

using machine learning, as per Section 5.4.2. Then, the metrics that we consider are the estimation error $\sum_{i\in[n]}\sum_{j\in[b]} z_{ij}\left|f_i^0 - \frac{\sum_{k\in[n]} z_{kj} f_k^0}{\sum_{k\in[n]} z_{kj}}\right|$, the similarity error $\sum_{i\in[n]}\sum_{j\in[b]} z_{ij}\sum_{k\in[n]} z_{kj}\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2$, and the overall error, i.e., the convex combination of the above two error terms, weighted by λ and $1-\lambda$, respectively, which is exactly the objective function that we use in the proposed formulation. We separately study the two error terms to shed light on the trade-off that the proposed approach is faced with. Moreover, we distinguish between the error on elements which appeared in the prefix (and hence their estimate is extracted from the learned hashing scheme) and the error on unseen elements which did not appear in the prefix (and hence their estimate is inferred using machine learning) to examine the individual performance of each component of the proposed approach. We also measure the running time (in seconds) of the algorithms (note that the running time includes the time to learn both the hashing scheme and the classifier).

Experiment 1: Impact of hyperparameter λ. In this experiment, we study the impact of the hyperparameter λ on the learned hashing scheme. We set $G = 6$ and run three different versions of `opt-hash`for varying λ: one that uses `milp` to learn the hashing scheme, one uses `bcd`, and one uses `dp`. We record the estimation, similarity, and overall error on the prefix, as well as the running time of each algorithm. To examine the degree of sub-optimality of `bcd`, we present the actual values of the error terms that constitute the objective function (estimation, similarity, and overall error), i.e., we do not convert them in a per element/per pair of elements scale, which would be more interpretable. The results are presented in Figure 5-4. The key takeaways from this experiment are as follows:

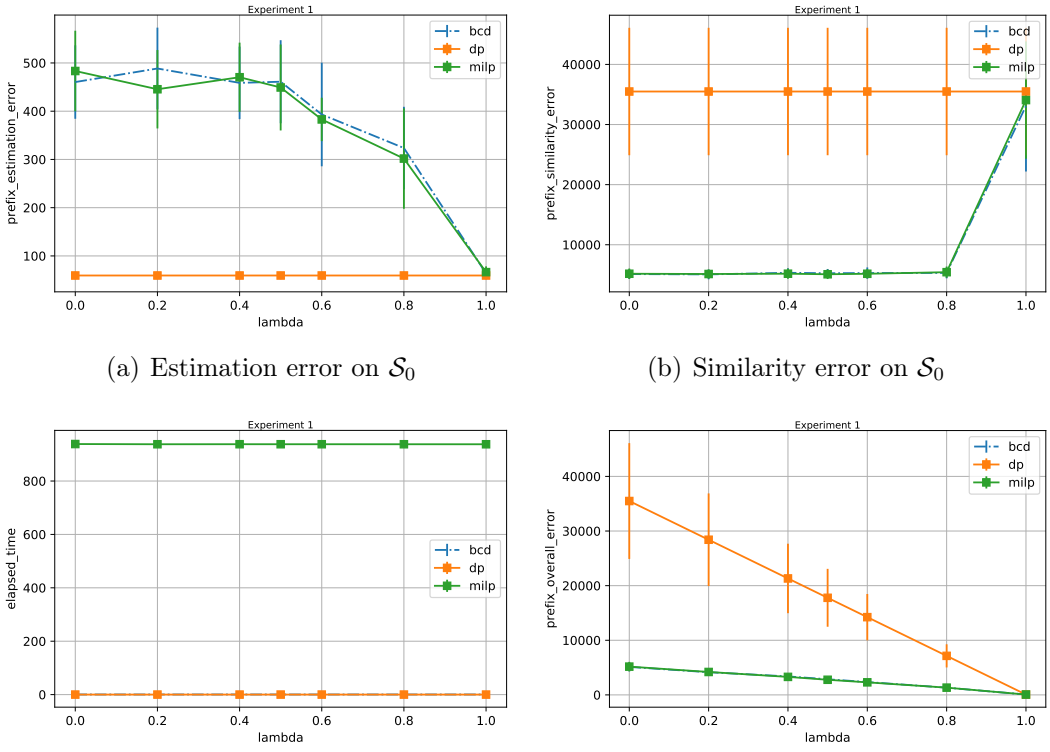
- `milp` achieves the smallest overall error at the cost of increased running times. Its edge over the heuristic `bcd` approach can be verified in terms of the estimation error, as it almost always improves over the solution obtained by `bcd`.

- The solutions obtained by `bcd` are of high quality; the improvement achieved by applying the exact `milp` approach is often negligible. For small problem sizes,

the runtime of `bcd` is less than a second.

- As expected, `dp` achieves the smallest estimation error, since it optimizes only for the estimation error independently of the value of λ. In terms of the similarity and the overall, the performance of `dp` is notably worse, whereas its running time is less than a second.

Note that, in the $\lambda = 1$ case, all three methods are able to find comparable near-optimal solutions. The small deviation is due to suboptimality tolerances of the algorithms used.



(a) Estimation error on $\mathcal{S}_0$

(b) Similarity error on $\mathcal{S}_0$

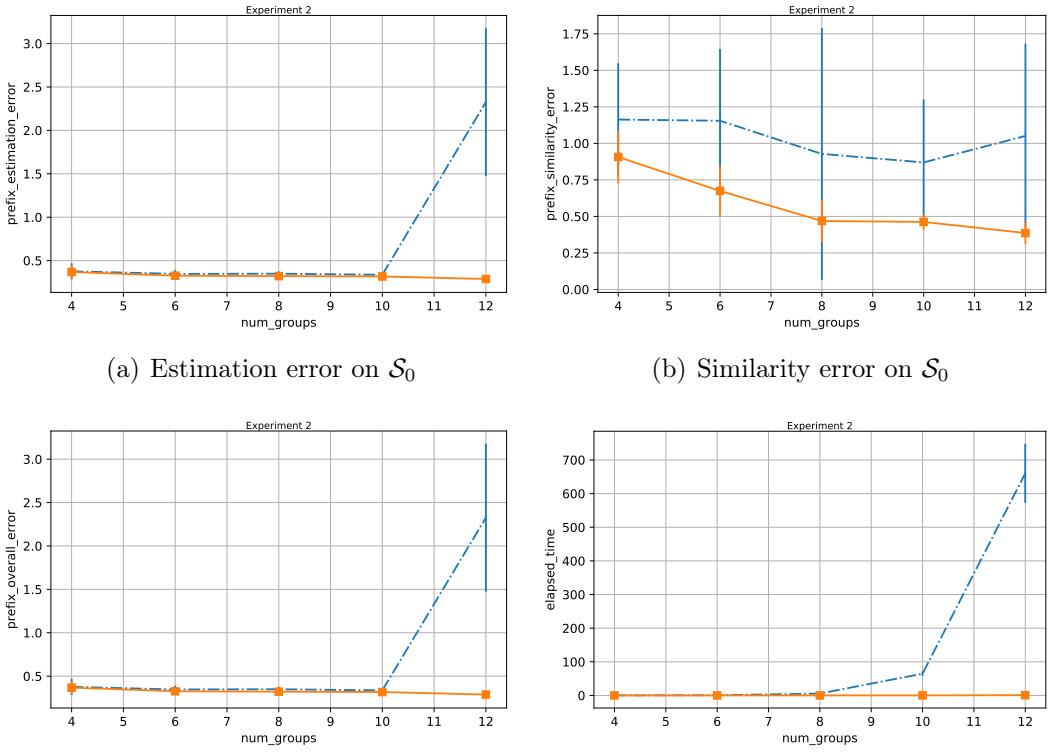
(c) Elapsed time (in sec)

(d) Overall error (objective function value) on $\mathcal{S}_0$

Figure 5-4: Impact of hyperparameter λ for $G = 6$.

Experiment 2: Comparison between `bcd` and `dp` in the $\lambda = 1$ case. In this experiment, we focus on the $\lambda = 1$ case and compare, for increasing values of G, `bcd` with `dp`; in this case, the latter is guaranteed to find the optimal hashing scheme. We

again record the estimation, similarity, and overall error on the prefix, as well as the running time of each algorithm. In this and in subsequent experiment, we convert the errors in a per element/per pair of elements scale. The results are presented in Figure 5-5. We observe that, for problems with $G \leq 10$, `bcd` computes near-optimal solutions fast; however, as G further increases, the performance of `bcd` deteriorates.



(a) Estimation error on $\mathcal{S}_0$

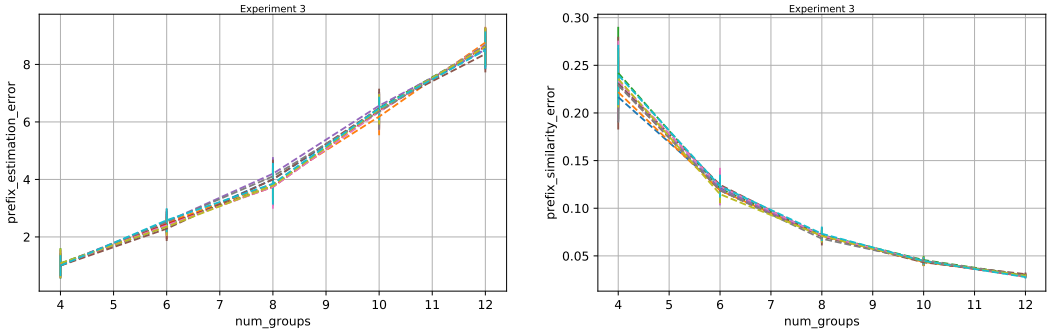
(b) Similarity error on $\mathcal{S}_0$

(c) Overall error (objective function value) on $\mathcal{S}_0$

(d) Elapsed time (in sec)

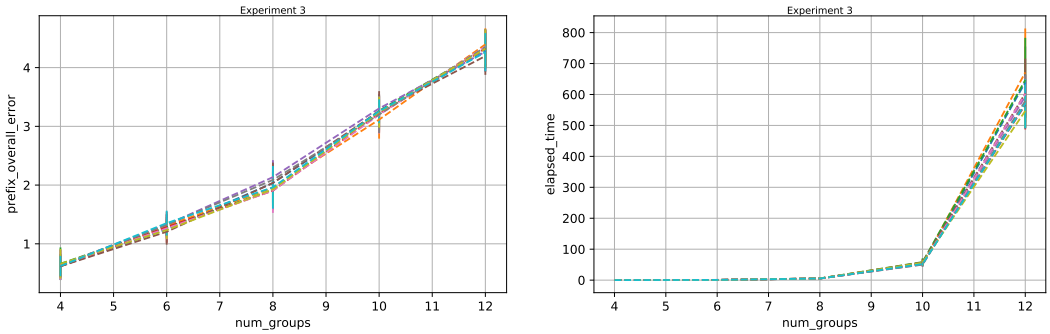
Figure 5-5: Comparison between `dp` and `bcd` for $\lambda = 1$.

Experiment 3: `bcd` from multiple starting points in the general λ case. In this experiment, we set $\lambda = 0.5$ and run `bcd` multiple times from different starting points and for increasing values of G to examine the stability of the solutions obtained. We again record the estimation, similarity, and overall error on the prefix, as well as the running time of each algorithm. The results, presented in Figure 5-6, indicate that `bcd` is robust to the (random) initialization of the algorithm and computes stable solutions.



(a) Estimation error on $\mathcal{S}_0$

(b) Similarity error on $\mathcal{S}_0$

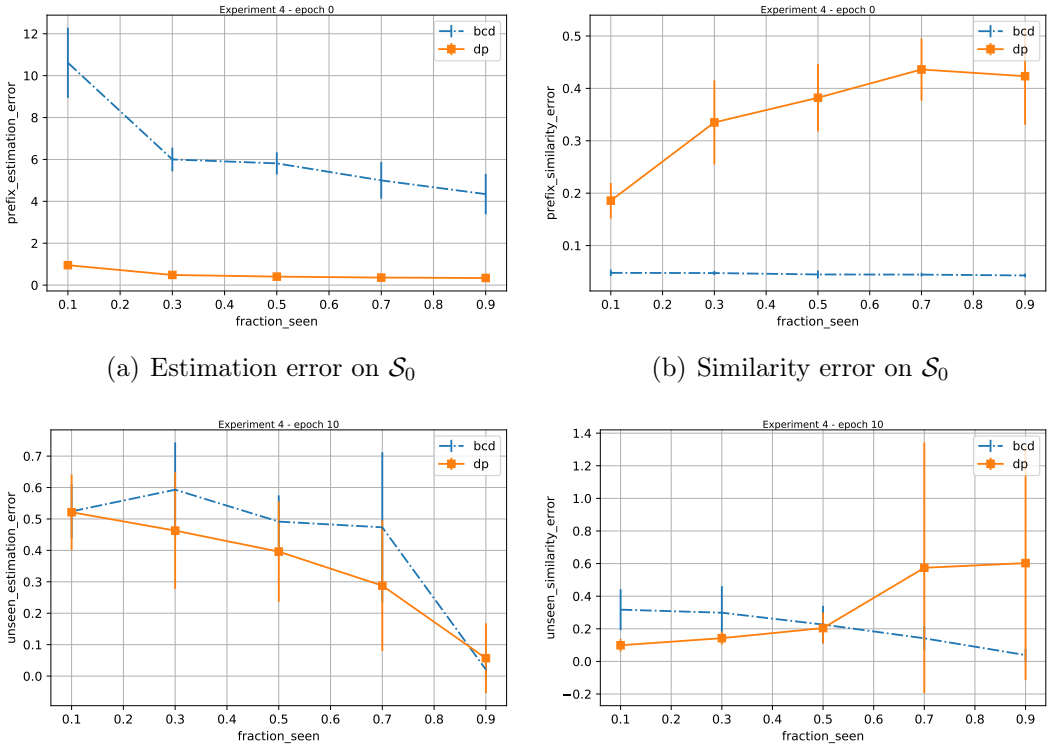


(c) Overall error (objective function value) on $\mathcal{S}_0$

(d) Elapsed time (in sec)

Figure 5-6: Comparison between `bcd` from multiple starting points for $\lambda = 0.5$.

Experiment 4: Impact of the fraction of elements seen in the prefix. In this experiment, we set $G = 10$ and vary the value of g_0, which controls the fraction of elements that appear in the prefix. We explore two approaches for learning the hashing scheme: first, we set $\lambda = 0.5$ and run `bcd`; then, we run `dp` (which implies $\lambda = 1$). We now record the estimation and similarity error both on the prefix $\mathcal{S}_0$ and on elements that did not appear in $\mathcal{S}_0$ but did appear within $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals after $\mathcal{S}_0$. Figure 5-7 suggests that observing more elements in the prefix results in a decrease of the estimation error on both seen and unseen elements at the cost of an increased similarity error.



(a) Estimation error on $\mathcal{S}_0$

(b) Similarity error on $\mathcal{S}_0$

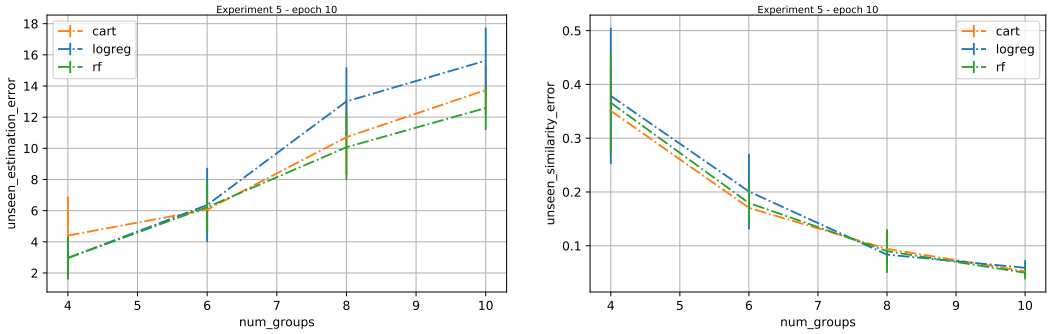
(c) Estimation error on elements $u \notin \mathcal{S}_0$ after $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals

(d) Similarity error on elements $u \notin \mathcal{S}_0$ after $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals

Figure 5-7: Impact of fraction of seen elements in the prefix (g_0) for $G = 10$.

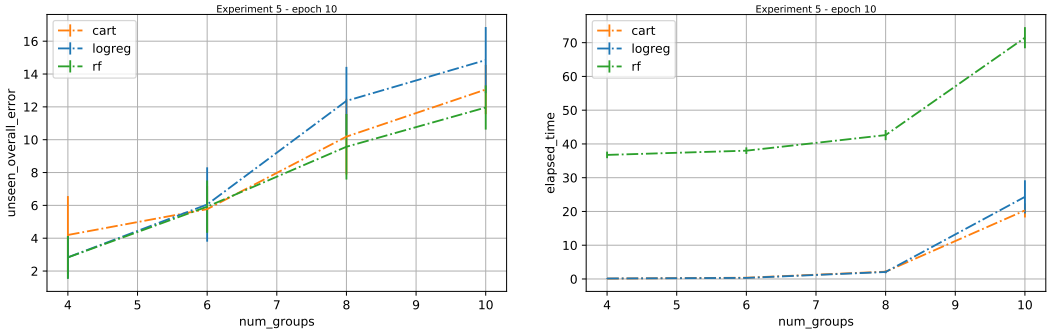
Experiment 5: Comparison between classification methods. In this experiment, we set $g_0 = 0.33$ and $\lambda = 0.5$, vary the value of G, and explore the impact

of using different types of classifiers (`logreg`, `cart`, `rf`) as part of `opt-hash`. We record the estimation, similarity, and overall error on elements that did not appear in $\mathcal{S}_0$ but did appear within $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals after $\mathcal{S}_0$. We also report the training time for each method. In Figure 5-8, we see that there is indeed merit in using non-linear classifiers. We remark, however, that the results heavily depend on the data generating process.



(a) Estimation error on elements $u \notin \mathcal{S}_0$ after $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals

(b) Similarity error on elements $u \notin \mathcal{S}_0$ after $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals



(c) Overall error (objective function value) on elements $u \notin \mathcal{S}_0$ after $|\mathcal{S}| = 10|\mathcal{S}_0|$ arrivals

(d) Elapsed time (in sec)

Figure 5-8: Comparison between classification methods.

5.6 Experiments on Real-World Data: Search Query Estimation

In this section, we empirically evaluate the proposed approach on real-world search query data. The task of search query frequency estimation seems particularly suited

for the proposed learning-based approach, given that popular search queries tend to appear consistently across multiple days.

5.6.1 Dataset

In the lines of [137], we use the AOL query log dataset, which consists of 21 million search queries (with 3.8 million unique ones) collected from 650 thousand anonymized users over 90 days in 2006. Each query is a search phrase in free text; for example, the 1^{st} most common query is "google" and appears 251,463 times over the entire 90-day period, the 10^{th} is "www.yahoo.com" and its frequency is 37,436, the 100^{th} is "mys" and its frequency is 5,237, the 1000^{th} is "sharon stone" and its frequency is 926, the 10000^{th} is "online casino" and its frequency is 146, and so forth. As shown in [137], the distribution of search query frequency indeed follows the Zipfian law and hence the setting seems ideal for their proposed algorithm (LCMS).

5.6.2 Baselines

As baselines, we use `count-min` and `heavy-hitter`. For each method, we maintain multiple versions corresponding to different values of the method's hyperparameters and report the best performing version. More specifically, for fixed sketch size (i.e., total number of buckets b), we report the best performing for `count-min`'s depth from the set $d \in \{1, 2, 4, 6\}$ and for `heavy-hitter`'s depth $d \in \{1, 2, 4, 6\}$ and number of heavy-hitter buckets $b_{heavy} \in \{10, 10^2, 10^3, 10^4\}$ (provided that b_{heavy} fits within the available memory, i.e., $b_{heavy} \leq b/2$). Additionally, we assume that `heavy-hitter` has access to an ideal heavy-hitter oracle, i.e., the IDs of the heavy-hitters in the test set (over the entire 90-day period) are known. Therefore, we compare the proposed method with the ideal version of the method proposed in [137], which was in fact shown to significantly outperform any realistically implementable version of `heavy-hitter` that relied upon non-ideal heavy-hitter oracles (e.g., recurrent neural network classifier).

5.6.3 Remarks on the Learned Hashing Scheme

As far as `opt-hash` is concerned, we make the following remarks:

- We consider the first day to be the observed stream prefix S_0 and use (part of) the queries $u \in \mathcal{U}'_0 \subseteq \mathcal{U}_0$ therein (along with their number of occurrences during the first day) to learn the optimal hashing scheme via Algorithm 6 and for $\lambda = 1$.

- The first day consists of over 200,000 unique queries and just storing their IDs would require 200,000 buckets. Thus, we randomly sample a subset of the observed queries, with probabilities proportional to their observed frequencies. We use the sampled subset of queries as input to Algorithm 6.

- For fixed number of total buckets b_{total}, we need to determine the ratio c between the number of buckets b that the learned hashing scheme will consist of and the number of queries n whose IDs we will store. Therefore, for user-specified b_{total} and c, we pick b and n according to

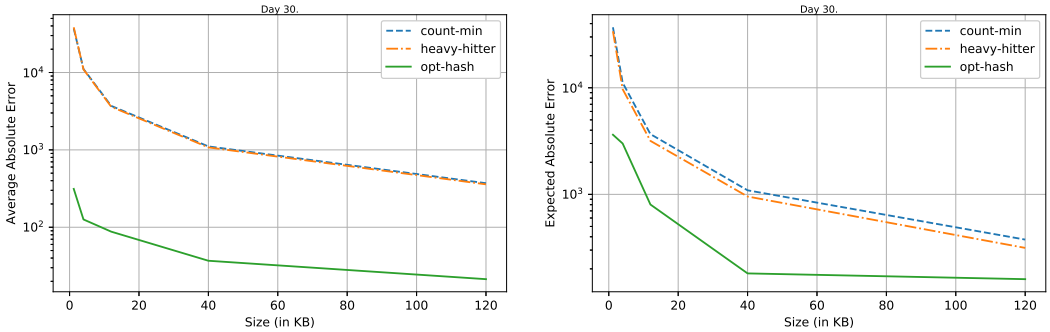
$$n = {}^{b_{\text{total}}}\!/_{1+c}, \quad b = b_{\text{total}} - n.$$

  In our experiments, we examine $c \in \{0.03, 0.3\}$.

- For the classifier $g : \mathcal{X} \to [b]$, mapping unseen queries $u \in \mathcal{U} \setminus \mathcal{U}'_0$ to buckets (as per Section 5.4.2), we found that `rf` achieves the best trade-off between training time and classification accuracy and use this model in the results we report.

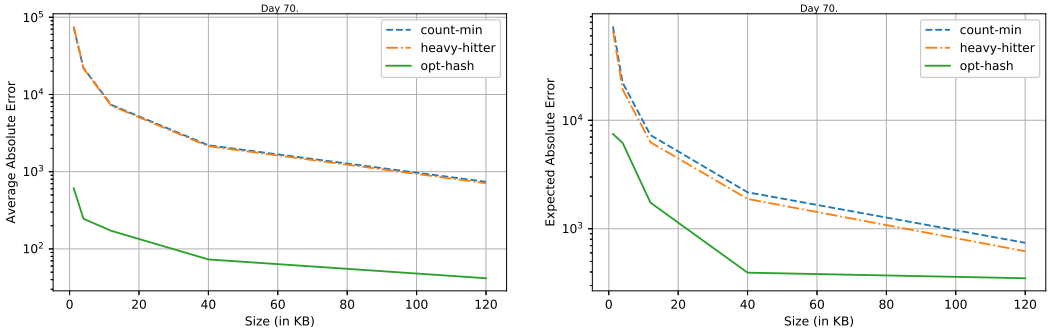
- To create input features for the classifier g, we follow a simple bag-of-words approach and only keep the 500 most common words in the training queries. We also include as features the number of ASCII characters in the query text, the number of punctuation marks, the number of dots, and the number of whitespaces. As a result, the proposed approach is simple and interpretable, yet strong (as we show next).

5.6.4 Results



(a) Average per element absolute error after the 30^{th} day

(b) Expected magnitude of absolute error after the 30^{th} day



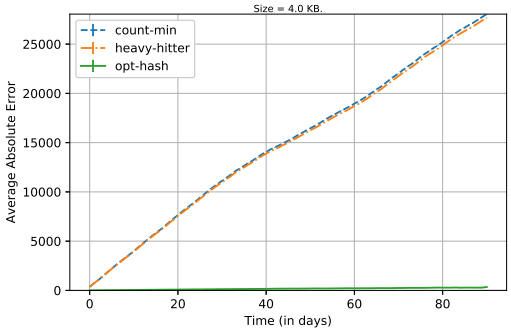
(c) Average per element absolute error after the 70^{th} day

(d) Expected magnitude of absolute error after the 70^{th} day

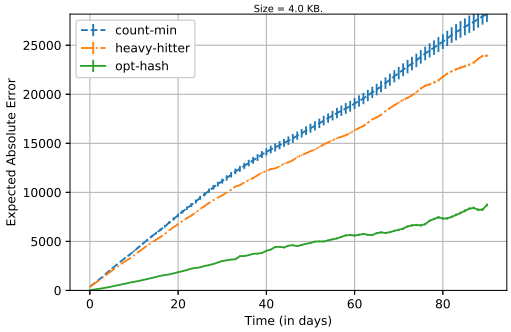
Figure 5-9: Estimation error as function of the estimator's size (in KB).

We implement our experiments in Python 3 and use the Scikit-learn machine learning package [183]. We independently repeat each experiment 5 times and report the averaged error, as well as its standard deviation. We remark that each bucket consumes 4 bytes of memory and hence the total number of buckets used in each experiment can be calculated as $b = \frac{m \cdot 10^3}{4}$, where m is the size of the estimator in KB. Moreover, we denote by $\mathcal{U}_t$ the set of queries that appear in day t, and by $\boldsymbol{f}^t_{\mathcal{U}_t}$ and $\tilde{\boldsymbol{f}}^t_{\mathcal{U}_t}$ their aggregated true frequencies and estimated frequencies, respectively, between days 0 and t.

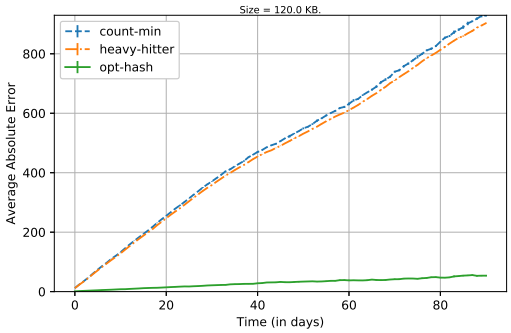
In Figure 5-9, we show the estimation error as function of the estimator's size in KB, after the 30^{th} and the 70^{th} day. On the the left (Figures 5-9(a) and 5-9(c)), we



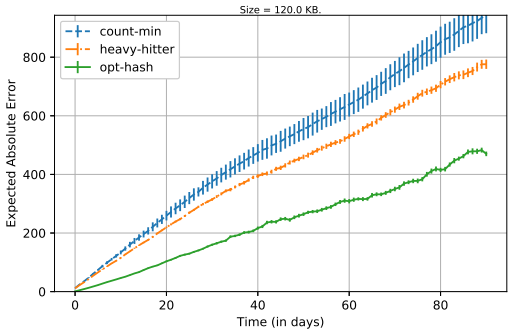
(a) Average per element absolute error using 4 KB of memory



(b) Expected magnitude of absolute error using 4 KB of memory



(c) Average per element absolute error using 120 KB of memory



(d) Expected magnitude of absolute error using 120 KB of memory

Figure 5-10: Estimation error as function of time (in days).

plot the average (per element) estimation error

$$\frac{1}{|\mathcal{U}_t|} \sum_{u \in \mathcal{U}_t} |f_u^t - \tilde{f}_u^t|.$$

On the the right (Figures 5-9(b) and 5-9(d)), we plot the expected magnitude of the absolute estimation error

$$\frac{1}{\sum_{u \in \mathcal{U}_t} f_u} \sum_{u \in \mathcal{U}_t} f_u^t \cdot |f_u^t - \tilde{f}_u^t|.$$

Notice that the former metric is expressed in a per element scale, that is, we normalize the overall error by the total number of elements $|\mathcal{U}_t|$ and hence all elements are penalized uniformly, whereas the second metric, the expected magnitude of the absolute estimation error, penalizes elements proportionally to their actual frequencies, as per Section 5.3.1.

We observe that the trend in the estimation error is very similar after the 30^{th} and the 70^{th} day. What changes is the absolute value of the estimation error, which, as expected, deteriorates with time, uniformly for all methods. The proposed method `opt-hash` consistently outperforms its competitors, in terms of both metrics. Unsurprisingly, as the size of all estimators increases, their errors drop. This is the case with both the average and the expected estimation error. We make the following additional remarks:

- The superiority of `opt-hash` is most notable in terms of average (per element) error. This is partly due to the fact that `opt-hash` does a substantially better job at estimating the frequencies of rarely occurring queries. In particular, queries that appear very few times are placed in the same bucket and hence the estimation error on them is small. In contrast, `heavy-hitter` and `count-min` often place such queries in the same bucket with queries of medium or even high frequencies, which produces big estimation error.

- The expected magnitude of the estimation error of `heavy-hitter` and `count-min` does seem to slowly converge towards that of `opt-hash` when the estimators' size

becomes sufficiently large. This indicates that `opt-hash` is particularly suited for low-space regimes and can achieve much more effective compression of the frequency vector.

- As far as `heavy-hitter` and `count-min` are concerned, the former does produce better estimates, which is in agreement with the results in [137]. The improvement is much more notable in terms of the expected magnitude of the estimation error. This observation is to be expected as well, given that `heavy-hitter` makes zero error on the most frequently occurring elements, which are heavily weighed in this metric.

Figure 5-10 reports the estimation error as function of time (in days), for two different memory configurations (4 KB in Figures 5-10(a) and 5-10(b), 120 KB in Figures 5-10(c) and 5-10(d)). The superiority of `opt-hash` is preserved over time, in terms of both metrics. Moreover, we observe `opt-hash` achieves the smallest standard deviation in its estimation error. This can be attributed to the fact that the mappings of elements to buckets are more stable than those of `heavy-hitter` and `count-min`, as they are obtained via optimization instead of randomization; the main source of randomness for `opt-hash` is the classifier.

We next experiment with memory configurations that vary between 1.2 KB and 120 KB, and compare `opt-hash` with `count-min` and `heavy-hitter`. The proposed approach provides an average improvement (over the entire 90-day period) by one to two orders of magnitude, in terms of its average (per element) absolute estimation error, and by 45-90%, in terms of its expected magnitude of estimation error. For example, with 120 KB of memory, `opt-hash` makes an average absolute estimation error of ~ 29 in estimating the frequency of each query, whereas the error of `heavy-hitter` is ~ 479 (Figure 5-9(a)). With 4 KB of memory, the errors of `opt-hash` and `heavy-hitter` are ~ 167 and $\sim 14,661$, respectively (Figure 5-9(c)). Table 5.2 shows the average (per element) error after the entire 90-day period as a percentage of each query's frequency for the 1^{st}, the 10^{th}, the 100^{th}, the $1,000^{th}$, and the $10,000^{th}$ most common queries.

An additional feature of `opt-hash` is that, by using interpretable features in

Query rank (by frequency)	Query frequency	Average error percentage (%)
1	251,463	0.01
10	37,436	0.08
100	5,237	0.55
1,000	926	3.13
10,000	146	19.86

Table 5.2: Average (per element) error as percentage of query's frequency.

its machine learning component, it provides insights into the underlying frequency estimation problem. In particular, the features that were consistently marked as most important are the four counts (i.e., number of ASCII characters in the query text, the number of punctuation marks, the number of dots, and the number of whitespaces), as well as the words "com," "www," "google," and "yahoo." Intuitively, this observation makes sense. For instance, a large number of ASCII characters and whitespaces would be indicative of a big query with multiple words, making it more likely to be rare. On the other hand, a query containing the word "google" would be more likely to be common, given that "google" is consistently part of the most frequently occurring queries.

5.7 Conclusions

In this chapter, we developed a novel approach for the problem of frequency estimation in data streams that relies on the use of optimization and machine learning on an observed stream prefix. First, we formulated and efficiently solved the problem of optimally (or near-optimally) hashing the elements seen in the prefix to buckets, hence providing a smart alternative to oblivious random hashing schemes. To this end, we reformulated the problem as a mixed-integer linear optimization problem, we developed an efficient block coordinate descent algorithm, and, in a special case, we used dynamic programming to solve the problem in linear time. Next, we trained a classifier mapping unseen elements to buckets. As we discussed, during stream processing, we only keep track of the frequencies of those elements that appeared in the prefix; the estimate of the frequency of any element (either seen or unseen) is

the average of the frequencies of all elements that map to the same bucket. We also described an adaptive approach that enables us to update the compressed frequency vector and keep track of the frequencies of all elements. We used synthetic data to investigate the performance, the scalability, and the impact of various design choices for the proposed approach; our study suggested that the proposed algorithms can compute optimal hashing schemes for problems with thousands of elements, using the mixed-integer linear optimization reformulation or the dynamic programming approach, and high quality hashing schemes for problems with tens of thousands of elements using the block coordinate descent algorithm. Finally, we applied the proposed approach to the problem of search query frequency estimation and evaluated it using real-world data and empirically showed that the proposed learning-based streaming frequency estimation algorithm achieves superior performance compared to existing streaming frequency estimation algorithms.

5.8 Technical Proofs

Proof of Lemma 12. We introduce variables $E \in \mathbb{R}^{n\times b}_{\geq 0}$ such that e_{ij} corresponds to the absolute estimation error associated with mapping element i to bucket j. Since we are minimizing a nonnegatively weighed sum of such nonnegative terms, it suffices to require that

$$e_{ij} \geq f^0_i - \frac{\sum_{k\in[n]} z_{kj} f^0_k}{\sum_{k\in[n]} z_{kj}}, \qquad e_{ij} \geq -f^0_i + \frac{\sum_{k\in[n]} z_{kj} f^0_k}{\sum_{k\in[n]} z_{kj}}, \tag{5.4}$$

for all $i \in [n], j \in [b]$. To get rid of the fractional term in (5.4), we multiply both equations with $\sum_{k\in[n]} z_{kj}$; this results in bilinear terms of the form $e_{ij} \sum_{k\in[n]} z_{kj}$. To linearize those, we introduce variables $\Theta \in \mathbb{R}^{n\times n\times b}_{\geq 0}$ such that $\theta_{ikj} = e_{ij} z_{kj}$ can be interpreted as the error associated with mapping element i to bucket j when k is also mapped therein. Since θ_{ikj} is the product of a binary variable and a continuous variable, we can linearize the constraint $\theta_{ikj} = e_{ij} z_{kj}$ by introducing a big-M constant

such that, for all $i \in [n], j \in [b]$, $e_{ij} \leq M$. We then require that

$$\theta_{ikj} \geq e_{ij} - M(1 - z_{kj}), \quad \theta_{ikj} \leq e_{ij}, \quad \theta_{ikj} \leq M z_{kj}, \tag{5.5}$$

for all $i \in [n], k \in [n], j \in [b]$. Thus, (5.4) can be rewritten as

$$\begin{aligned} \sum_{k \in [n]} \theta_{ikj} &\geq f_i^0 \sum_{k \in [n]} z_{kj} - \sum_{k \in [n]} f_k^0 z_{kj}, \\ \sum_{k \in [n]} \theta_{ikj} &\geq -f_i^0 \sum_{k \in [n]} z_{kj} + \sum_{k \in [n]} f_k^0 z_{kj}, \end{aligned} \tag{5.6}$$

which is linear in all variables. To linearize the other bilinear term that appears in the objective function, we introduce another set of auxiliary variables $\Delta \in [0, 1]^{n \times n \times b}$ such that $\delta_{ikj} = z_{ij} z_{kj}$ indicates whether elements i and k are mapped together to bucket j. We then have the constraints

$$\delta_{ikj} \geq z_{ij} + z_{kj} - 1, \quad \delta_{ikj} \leq z_{ij}, \quad \delta_{ikj} \leq z_{kj}, \tag{5.7}$$

for all $i \in [n], k \in [n], j \in [b]$. Using the above new variables, the objective function can be written as

$$\sum_{i \in [n]} \sum_{j \in [b]} \left[\lambda \theta_{iij} + (1 - \lambda) \sum_{k \in [n]} \delta_{ikj} \| \boldsymbol{x}_i - \boldsymbol{x}_k \|^2 \right]. \tag{5.8}$$

Finally, we have to properly select the constant M in (5.5) so that it is a valid upper bound for the variables E; such a bound can be obtained by setting $M \geq \max_{i \in [n]} f_i^0$, i.e., the estimation error associated with any element cannot be greater than the largest frequency observed in the prefix. □

Chapter 6

Solar Capacity Expansion at OCP under Variability and Volume

6.1 Introduction

On December 12, 2015, 195 countries met at COP-21 in Paris and signed a climate agreement that aims to limit average global temperature increases to within 2 degrees Celsius—and preferably within 1.5 degrees Celsius—of preindustrial levels by the end of the 21st century [86]. As 1.1 degrees of global temperature rises have already occurred, this pledge will be implemented by rapidly decarbonizing industrialized nations and subsequently decarbonizing the rest of the world. To minimize hardship during this transition, all participants in the global economy should decarbonize wherever doing so is profitable as soon as possible. The Kingdom of Morocco is a signatory to the Paris agreement, and in 2021 it proposed to reduce its carbon emissions by 45.5% by 2030 [91].

The OCP group (formerly Office Cherifien des Phosphates) constitutes a significant 5.6% of Morocco's GDP [116] and is responsible for managing Morocco's 70% share of the world's phosphate rock reserves to manufacture fertilizer [206]. To help implement Morocco's green energy pledge, OCP recently announced a green initiative to partially decarbonize its production process by investing in solar panels and batteries, thereby reducing its overall carbon footprint by 50% by 2030 (compared to 2014 levels) and

be carbon neutral by 2040 [177]. This commitment is significant for two reasons. First, as of 2020, 57% of OCP's energy needs are satisfied via non-renewable resources, and OCP is interested in dramatically reducing this portion. Second, fertilizer is responsible for around $30\% - 50\%$ of the world's food production [204], and with a growing global population and increasingly protein-rich dietary habits, it is desirable to produce food in as sustainable a fashion as possible.

In this chapter, we describe a robust optimization (RO) methodology that OCP is currently using to size its investment and partially implement its decarbonization pledge by installing a judicious mix of solar panels and batteries throughout its production system. We also propose techniques which allow OCP to successfully operate its system in the presence of uncertainty induced by intermittent solar generation, particularly concerning deciding when to release or store energy in batteries. By co-optimizing the cost of installing renewable energy and the cost of procuring electricity from the Moroccan national grid, we drastically curtail the carbon emissions which arise due to OCP's energy needs. We remark that there are other sources of emissions in OCP's production process that we do not address, and thus decarbonizing OCP's electricity supply does not fully decarbonize OCP; see [26] for an overview of OCP's production process.

Our model forecasts that decarbonizing OCP's electricity supply is actually profitable for OCP due to the abundance of solar energy in Morocco, relatively high energy prices, and relatively low real interest rates. This forecast has significant managerial implications; while companies and nation-states recognize decarbonization as a laudable goal—indeed a profitable one for the global economy as a whole [5]—they sometimes view it as an expensive luxury that members of the developing world cannot afford while industrializing their economies. Indeed, the Paris climate agreement reflects this belief, by requiring that wealthier industrialized nations decarbonize sooner since they can afford to do so. However, we demonstrate by example that decarbonization can (at least sometimes) be profitable; therefore, access to global capital markets and low regulatory barriers to installing renewables may sometimes be sufficient to decarbonize.

Ultimately, whether decarbonization is profitable for a large industrial consumer depends on the amount of solar capacity available, the price of solar panels and batteries, local energy prices, and interest rates. Therefore, we hope that our methodology can be applied elsewhere to ascertain whether decarbonization via solar panels (or other technologies) is profitable for a given consumer.

6.1.1 An Overview of OCP's Production Process

In this section, we provide an overview of OCP's current manufacturing process and its energy consumption behavior (see [177] for further details), with a view to model and eventually decarbonize the process. Figure 6-1 summarizes OCP's current process visual-spatially.

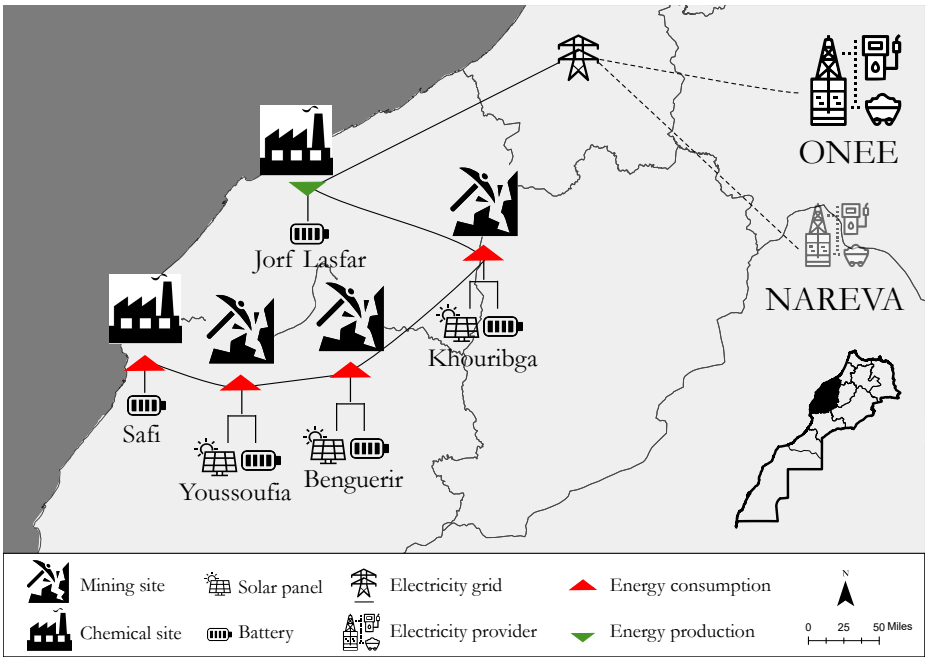


Figure 6-1: OCP's manufacturing and phosphate rock mining sites across Morocco, and its energy suppliers.

Current Production Process (as of 2020). Each year, OCP extracts around 40 million tons of raw phosphate rock from eight mining sites. This extracted rock is first enriched at washing facilities, and then either exported via a port (around 25% of extracted phosphate rock) or transported to one of two chemical sites to

undergo further processing. A portion of the phosphate rock that arrives at the two processing platforms is combined with sulfuric acid to produce about seven million tons of phosphoric acid, which is then directly exported to be used by a variety of economic sectors, including the food and pharmaceutical industries. The remainder is processed with ammonia to produce about 12 million tons of fertilizers. All in all, OCP supplies more than half of all phosphate-based products sold in Africa.

Energy Consumption Behavior. OCP's energy needs are currently satisfied via carbon-emitting sources, with a small minority of its needs met via wind energy. Specifically, its demand is currently met by a combination of: wind energy (6%), which OCP procures via power purchase agreements, cogeneration (37%), which is generated by recovering waste heat released during the sulfuric acid production within OCP's processing sites, industrial fuel (35%), diesel (9%), natural gas (4%), and electricity purchased from the grid (9%). Excluding cogeneration, around 89.5% of OCP's demand is currently satisfied via carbon-emitting sources.

Gentailers. Excluding cogeneration, OCP purchases more than 60% of the electricity which it uses. OCP's electricity providers are the Office National de l'Electricité et de l'Eau Potable (ONEE), the national electricity company and leading operator in the field of electricity in Morocco, and Nareva, a private energy company. ONEE has a global installed net generation capacity of 11 GW and generates around 34% of its energy via renewable sources. Nareva has an installed capacity of about 2 GW, and generates around 37% of its energy via renewable sources.

In this chapter, we develop a methodology which reduces the amount of carbon-based electricity OCP procures from the grid by installing solar panels and batteries across OCP's system. Our methodology explicitly models (a) a complex of four mining sites in Khouribga, (b) the mining sites in Benguerir and (c) Youssoufia, as well as OCP's two chemical sites, located in (d) Jorf and (e) Safi. Due to physical constraints, solar panels can be installed only at mining sites, whereas batteries can be installed at all sites. All sites are connected to the grid; however, the OCP-Nareva contract

allows purchasing electricity from Nareva only at mining sites. We now review the relevant literature on this topic (Section 6.1.2), before stating our full contributions explicitly (Section 6.1.3).

6.1.2 Relevant Literature

Our work arises at the intersection of two related areas of the Analytics and Operations Management literature: (a) techniques for optimally expanding a production system's capacity and (b) data-driven methods for solving multi-stage RO problems. We review both areas.

Generator Capacity Expansion Literature. The problem of expanding a generator's capacity is one of the most frequently studied in the power systems literature. Classically, generators rank their options according to their long-run marginal costs using a screening curve (see, e.g., [162, 205]), which, as discussed by [105] corresponds to solving a linear optimization problem to determine which generators should expand at each location.

Unfortunately, applying this approach out of the box can be inaccurate, since it assumes future supply and demand are deterministic. In reality, wind and solar generation are intermittent and could be considered as part of a future generation mix even when not currently present. Moreover, future consumer demand is uncertain since it depends on local weather conditions and population growth among other factors. As reflected in stochastic [57, 199] and robust [27, 39] optimization textbooks, screening curves are therefore highly suboptimal in the presence of uncertainty.

To more rigorously address capacity expansion, several authors have proposed models that explicitly account for uncertainty via stochastic or robust optimization. Among others, [11] model capacity expansion problems as multi-stage stochastic integer programs and approximately solve them via their linear relaxations; [10] solve capacity expansion problems to global optimality via branch-and-bound; [201] solve them in a more scalable manner via Dantzig-Wolfe decomposition; while [241] apply a very efficient adaptive optimization scheme; see [111, 122] for reviews.

More recently, capacity expansion models have also been developed for regulators aiming to address sustainability concerns by decarbonizing electricity markets. Among others, [60] proposes a generation and transmission capacity expansion model for decarbonizing the ERCOT (Texas) electricity market by installing additional wind and solar capacity. In a related direction, [106] recently developed a model for decarbonizing the New Zealand Electricity Market while accounting for its hydro-dominated nature. Both models either constrain the amount of non-renewable energy consumed explicitly, or equivalently (from a Lagrangian perspective) impose a carbon price to reduce carbon emissions.

In this work, we take a different perspective on capacity expansion. Rather than considering the perspective of an individual generator that sells its production to consumers or a regulator aiming to reduce emissions across an entire electricity system, we take the perspective of a vertically integrated generator-consumer pair, or prosumer, which produces electricity to satisfy its own demand. Our model is also designed in collaboration with the prosumer, thus making it arguably more physically realistic than existing models in the literature.

Data-driven methods for multi-stage RO. The idea of planning over time by partitioning decision variables across multiple stages has been popular since it was originally proposed by George Dantzig and Evelyn Beale in 1955, eventually giving rise to the paradigm of multistage stochastic optimization [85, 24]. In this paradigm, at each stage, a decision maker optimizes a subset of the decision variables while knowing the past deterministically and the future stochastically. In response, nature selects some of the uncertain parameters from a (known) probability distribution. In practice, inferring nature's joint probability distribution requires a prohibitive amount of historical data and therefore optimizers replace the true underlying distribution with its empirical distribution. This technique is known as sample average approximation (SAA) (c.f. [199] Chapter 5). In the single-stage case, given a sample $\mathcal{S} = \{\xi_1, \xi_2, \ldots, \xi_N\}$ of a random variable $\xi(\omega)$ defined on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, an optimizer approximately minimizes the expected value of a cost function $c(\boldsymbol{x}, \boldsymbol{\xi})$, $\mathbb{E}[c(\boldsymbol{x}, \boldsymbol{\xi})]$

over a convex set $\mathcal{X}$ by solving $\min_{\boldsymbol{x}\in\mathcal{X}} \frac{1}{N}\sum_{i=1}^{n} c(\boldsymbol{x}, \boldsymbol{\xi}_i)$. Owing to the success of techniques like SAA and related multi-stage algorithms like Stochastic Dual Dynamic Programming [184], stochastic optimization is frequently and sometimes successfully used to address important problems in energy, financial and logistics.

Unfortunately, stochastic optimization has three main drawbacks. First, without making potentially heroic assumptions on the structure of a problem, it is computationally intractable. Indeed, [127] have shown that solving a two-stage stochastic optimization problem exactly is $\#P$-hard, i.e., as hard as counting the number of optimal solutions to a binary quadratic problem, while sample-based methods require a number of scenarios exponential in the number of stages [198]. Second, stochastic optimization assumes that uncertain parameters can be drawn from a (known) probability distribution, when in reality optimizers usually only have access to a limited amount of historical data, and estimating a potentially high-dimensional distribution from this data can introduce significant estimation error. Third, SAA tends to generate policies which overfit the empirical distribution and perform poorly out-of-sample [214].

To avoid the computational intractability of stochastic optimization, several authors [203, 28, 52] designed an alternative modeling paradigm called RO, which plans under uncertainty by designing policies which perform best under the worst-case parameter realization contained within an uncertainty set. Crucially, if uncertainty sets are designed appropriately, RO problems can be reformulated as deterministic equivalents of a comparable size and similar complexity as the original nominal problem [53]. Moreover, as discussed in [21], one can use uncertainty sets that are derived from asymptotic implications of probability theory, and therefore obtain tangible probabilistic guarantees on the feasibility of the solution to the RO problem. A frequent critique of RO is that a suboptimal solution to a stochastic optimization problem may in fact perform better in practice than an optimal solution to a robust one, because RO aims to generate solutions with good worst-case performance and thus is too conservative in circumstances where we are interested in optimizing average-case performance and uncertainty can be amortized over time.

To reduce the conservatism of RO while retaining its tractability, several authors [88,

220] propose a related modeling paradigm called distributionally robust optimization (DRO), which unifies stochastic optimization with RO by optimizing for the worst-case measure over an ambiguity set of probability measures. Surprisingly, if the ambiguity set is specified appropriately then this approach remains tractable, due to the duality between measures and moments (c.f. [51, 182]). Moreover, unlike in the stochastic case (which typically overfits), or the robust case (which typically underfits), one can easily generalize DRO to account for the fact that we work with historical data rather than probability distributions without sacrificing much out-of-sample performance. Indeed, as established by [214], a variant of SAA where we pessimize over all distributions within a given distance from the empirical one, i.e., solve $\min_{\boldsymbol{x}\in\mathcal{X}} \sup_{\mathbb{Q}\in\mathcal{P}_\delta} \sum_{i=1}^{N} \mathbb{Q}(\omega^i) c(\boldsymbol{x}, \boldsymbol{\xi}^i)$, where $\mathcal{P}_\delta$ is the set of all measures close to the empirical measure and with the same support, guarantees that our out-of-sample disappointment will never be too high.

6.1.3 Contributions and Outline

The key contributions of the chapter are twofold. First, we present an optimization *methodology* that guides OCP's investment in renewable energy and prescribes its operational strategy over the next 20 years. Via RO, the proposed model guards against the uncertainty associated with solar generation on a day-to-day basis. Moreover, via the use of DRO, the model guards against changes in seasonal generation output induced by shifting weather patterns as the Moroccan climate changes with increasing atmospheric CO_2 levels. Furthermore, the model is data-driven: it combines optimization with machine learning (ML) to learn from simulated data while accounting for prediction inaccuracies. Second, *from an implementation perspective*, our approach facilitates a significant (and profitable) investment in renewable energy. As of our initial submission, OCP has implemented our model and is using it to size its investment, which it expects to be at least ten billion MAD (approx. one billion USD) according to an optimal solution generated by our methodology. We remark that our methodology supports a long-term investment and therefore the full implications of our implementation will emerge over time as OCP physically installs solar panels and

batteries. In terms of immediate impact, our model has already demonstrated its value as *a powerful decision support tool which OCP has used to evaluate a suite of investment options*.

Technical Contributions. From a technical standpoint, (i) we develop a multi-period SAA formulation to co-optimize OCP's investment and operational decisions in a tractable fashion. To tackle the large dimensionality of the problem, we propose a novel, ML-based scenario reduction technique, which allows us to plan for a reduced set of "typical days" rather than all days in the planning horizon. (ii) We use RO to account for variability in daily solar capacities (uncertainty in the weather). We propose a novel, data-driven RO approach that constructs uncertainty sets around the amount of solar power available in a given day to reflect the idea that at the start of a day a weather forecast is not perfect. This gives rise to a notion of averaging over uncertainty sets and helps prevent conservatism. (iii) We use DRO to protect against variability in seasonal weather patterns induced by climate change (uncertainty in the climate).

As a philosophical remark on the use of RO vs DRO: to our knowledge, there has not been work that uses both RO and DRO in the same problem. Our modeling choice of RO vs DRO follows from the amount of data we have. We use considerable amounts of historical data to inform the uncertainty in solar availability that RO guards against, but only a small amount of data to model ambiguity in seasonal weather patterns induced by climate change.

Managerial Implications. Our results provide valuable insights for large electricity consumers seeking to reduce their carbon emissions or energy-related expenses. Our methodology demonstrates that whether decarbonizing via installing solar panels and batteries is profitable is primarily a function of local solar capacity factors, the cost of borrowing capital, and current energy prices. Therefore, although we focus on decarbonizing a fertilizer manufacturer's electricity supply, our approach could be useful in other contexts, such as manufacturing steel, cement, or chapter.

Outline. The rest of this chapter is laid out as follows. In Section 6.2, we lay out OCP's full multi-period SAA problem which allows OCP to co-optimize its investment and operational decisions. In Section 6.3, we develop a data-driven methodology to robustify our model against uncertainty in both day-to-day weather and seasonal climate patterns, via a combination of RO and DRO. In Section 6.4, we study the impact of our model and demonstrate that installing solar panels and batteries is profitable if energy costs are sufficiently high.

6.2 A Sample Average Approximation Model

In this section, we lay out a mathematical optimization model which optimizes OCP's green energy strategy, by making optimal *strategic* decisions regarding where it should build new solar capacity and batteries, and optimal *operational* decisions, which allow it to take advantage of its new assets. The key assumption in this section is that the amount of solar energy at a given site varies over time in a deterministically knowable way; we relax this assumption in Section 6.3.

We first provide an overview of (Section 6.2.1) and formulate (Section 6.2.2) our sample-average model, which optimizes both strategic and operational decisions at the start of the planning horizon given an empirical distribution of solar capacity factors. Next, we develop a highly scalable real-time model which allows OCP to make real-time operating decisions for the next 24 hours after receiving a solar capacity factor forecast (Section 6.2.3). Finally, we delve deeper into the data and the ML methodology we used to derive a reduced set of scenarios corresponding to solar capacity factors (Section 6.2.4). All-in-all, we provide a highly accurate yet tractable approximation of OCP's full multi-period planning problem, which eventually guides its investment decisions.

6.2.1 Model Description

Our overall multi-period model takes as inputs (i) the costs of procuring OCP's strategic assets, e.g., the cost of procuring and installing one kW of solar panels or one

kWh of battery storage, OCP's internal cost of capital, and the salvage value of one kWh of batteries or one kW of solar panels at the end of the planning horizon, (ii) the time-dependent costs of operating OCP's system, and (iii) other information required to operate OCP's system on a day-to-day basis. It then provides as outputs (i) optimal *strategic* decisions regarding where OCP should build new solar capacity and batteries at the start of each year, and (ii) an optimal *policy* which allows OCP to operationalize its assets in each period of the planning horizon. Our assumption that strategic assets are only installable at the start of each year is for computational tractability: we could consider installing assets at, e.g., the start of each month, although this would not significantly change the policies computed by the model while reducing its tractability.

6.2.2 Model Formulation

We now formulate the capacity expansion model as a multi-period linear optimization problem, which we refer to as the sample average approximation (SAA) model. Table 6.1 lists the parameters used throughout the model, which we set in collaboration with the OCP team. Indices that correspond to time periods are written as superscripts, and indices that correspond to locations or different electricity providers are written as subscripts.

Sets and Indices. In the capacity expansion planning model, we are given a set of nodes $\mathcal{N}$ corresponding to mines and chemical factories which OCP operates, a set of arcs $\mathcal{A}$ corresponding to pairs of nodes which are connected via transmission lines, a set of hours $\mathcal{H}$, days $\mathcal{D}$, and months $\mathcal{M}$ which OCP operates its system over, and a set of years $\mathcal{Y}$ which OCP optimizes investment decisions over. Given these sets, OCP decides which strategic assets to procure, when it should procure them, and how it will operationalize them over different hours, months, and years.

Owing to the nature of the Moroccan national grid, OCP has the option of both purchasing electricity at a given node n or purchasing it at a different node n' and then "renting" a line (or set of lines) connecting n and n'. In this case, OCP pays the marginal price of procuring electricity at node n', plus a transmission price for

transmitting energy from node n' to node n.

Reduced scenarios. To reduce the dimensionality of the problem while treating it in a data-driven fashion, we assume that the set $\mathcal{D}$ contains a reduced set of scenarios which occur with certain probability in each month, rather than all days in the planning horizon (typically $|D| = 10$). Each such reduced scenario can be loosely viewed as one of a few "typical days" that can occur throughout the year. Each scenario $d \in \mathcal{D}$ occurs with probability $P^{d,m,y}$ in month $m \in \mathcal{M}$ in year $y \in \mathcal{Y}$. In Section 6.2.4, we present the methodology we apply to obtain the scenarios from the available data sources.

Decision Variables. We now lay out the strategic decision variables optimized by the model, followed by the operational decisions, which are made on an hourly basis.

Strategic Decision Variables. The key strategic decision variables are:

$b_n^y \in \mathbb{R}_+$: The number of kWh of battery storage built at node $n \in \mathcal{N}$ in year $y \in \mathcal{Y}$; (6.1a)

$z_n^y \in \mathbb{R}_+$: The number of kW of solar panels built at node $n \in \mathcal{N}$ in year $y \in \mathcal{Y}$. (6.1b)

Operational Decision Variables. The key operational decision variables we use are:

$f_a^{h,d,m,y}$: The DC load-flow (in kW) through line $a \in \mathcal{A}$ in hour $h \in \mathcal{H}$ of day $d \in \mathcal{D}$ of month $m \in \mathcal{M}$ of year $y \in \mathcal{Y}$; (6.2a)

$s_n^{h,d,m,y}$: Energy stored (in kWh) in batteries at node $n \in \mathcal{N}$ in hour $h \in \mathcal{H}$ of day $d \in \mathcal{D}$ of month $m \in \mathcal{M}$ of year $y \in \mathcal{Y}$; (6.2b)

$r_n^{h,d,m,y}$: Power (in kW) discharged at node $n \in \mathcal{N}$ in hour $h \in \mathcal{H}$ of day $d \in \mathcal{D}$ month $m \in \mathcal{M}$ year $y \in \mathcal{Y}$; (6.2c)

Table 6.1: Summary of notation. Calligraphic letters refer to sets, Roman/Greek letters refer to problem data.

Symbol	Description
General Setting	
$\mathcal{H}$	Set of hours in each day, $\{1, \ldots, 24\}$
$\mathcal{D}$	Set of reduced scenarios
$\mathcal{M}$	Set of months in a calendar year, $\{1, \ldots, 12\}$
$\mathcal{Y}$	Set of years in OCP's planning horizon; i.e., $\{1, \ldots, 20\}$
$\mathcal{N}$	Set of nodes in the network, i.e., {Jorf, Safi, Benguerir, Youssoufia, Khouribga}
$\mathcal{A}$	Set of all arcs in the network
$D^{m,y}$	Number of days in month $m \in \mathcal{M}$ in year $y \in \mathcal{Y}$
$P^{d,m,y}$	Probability of reduced scenarios of type $d \in \mathcal{D}$ in month $m \in \mathcal{M}$ in year $y \in \mathcal{Y}$
Investment Decisions	
B	Investment budget in MAD (Moroccan dirham)
ρ	Discount factor, i.e., 0.96
c_b^y	Cost of purchasing and installing one kWh of batteries in year $y \in \mathcal{Y}$
c_s^y	Cost of purchasing and installing one kW DC of solar panels in year $y \in \mathcal{Y}$
OCP Operations	
Operational Data	
R	Constant which converts energy released from batteries into a rate, i.e., 1
ξ	Annual rate of solar generation capacity degradation, i.e., 0.995
ν	Annual rate of battery storage degradation, i.e., 0.96
ψ	Proportion of energy stored in a battery available an hour later, i.e., 0.95
β	Fraction of daily amount of power produced by solar panels that may be sold, i.e., 0.2
$\mathcal{I}(n)$	Set of arcs $a = (i, n)$ flowing into node $n \in \mathcal{N}$
$\mathcal{O}(n)$	Set of arcs $a = (n, i)$ flowing out of node $n \in \mathcal{N}$
K_a	Capacity limit in kW on the flow through arc $a \in \mathcal{A}$
η_a	The transmission efficiency coefficient for arc $a \in \mathcal{A}$, i.e., $\eta = 0.99$
Time-Dependent Data	
G_o^h	ONEE generation capacity in kW at node $n \in \mathcal{N}$ in hour $h \in \mathcal{H}$
G_n^h	NAREVA generation capacity in kW at node $n \in \mathcal{N}$ in hour $h \in \mathcal{H}$
$v^{h,d}$	Capacity factor for a solar panel in hour $h \in \mathcal{H}$ of reduced scenario $d \in \mathcal{D}$
$p_{O,n}^{h,m}$	Marginal cost of energy in MAD/kWh from ONEE at node $n \in \mathcal{N}$ at time $h \in \mathcal{H}$, $m \in \mathcal{M}$
$p_{N,n}^{h,m}$	Marginal cost of energy in MAD/kWh from NAREVA at node $n \in \mathcal{N}$ at time $h \in \mathcal{H}$, $m \in \mathcal{M}$
$p_{w,n}^{h,m}$	Marginal feed-in price in MAD/kWh for selling electricity at node $n \in \mathcal{N}$ at time $h \in \mathcal{H}$, $m \in \mathcal{M}$
$c_{r,a}^{h,m}$	Marginal cost in MAD/kWh of renting line $a \in \mathcal{A}$ at time $h \in \mathcal{H}$, $m \in \mathcal{M}$
$d_n^{h,d,m,y}$	Aggregate demand in kWh at node $n \in \mathcal{N}$ at time $h \in \mathcal{H}$, $d \in \mathcal{D}$, $m \in \mathcal{M}$, $y \in \mathcal{Y}$

$x_{O,n}^{h,d,m,y}$: Power procured (in kW) from grid from ONEE at node (6.2d)

$n \in \mathcal{N}$ in hour $h \in \mathcal{H}$ of day $d \in \mathcal{D}$ month $m \in \mathcal{M}$ year $y \in \mathcal{Y}$;

$x_{N,n}^{h,d,m,y}$: Power procured (in kW) from grid from NAREVA at node (6.2e)

$n \in \mathcal{N}$ in hour $h \in \mathcal{H}$ of day $d \in \mathcal{D}$ month $m \in \mathcal{M}$ year $y \in \mathcal{Y}$;

$w_n^{h,d,m,y}$: Power sold (in kW) to grid at node (6.2f)

$n \in \mathcal{N}$ in hour $h \in \mathcal{H}$ of day $d \in \mathcal{D}$ month $m \in \mathcal{M}$ year $y \in \mathcal{Y}$;

Optimizing the above decision variables using the problem data in Table 6.1 yields the problem:

$$\min \quad \sum_{y\in\mathcal{Y}} \Bigg[\underbrace{\sum_{n\in\mathcal{N}} c_b^y(\rho^y - \rho^{|\mathcal{Y}|})b_n^y}_{\text{cost of batteries}} + \underbrace{\sum_{n\in\mathcal{N}} c_s^y(\rho^y - \rho^{|\mathcal{Y}|})z_n^y}_{\text{cost of solar}} + \underbrace{\sum_{a,m,d,h} \rho^y D^{m,y} P^{d,m,y} c_{r,a}^{h,m} |f_a^{h,d,m,y}|}_{\text{cost to rent lines}}$$

$$+ \underbrace{\sum_{n,m,d,h} \rho^y D^{m,y} P^{d,m,y} \left(p_{O,n}^{h,m} x_{O,n}^{h,d,m,y} + p_{N,n}^{h,m} x_{N,n}^{h,d,m,y} - p_{w,n}^{h,m} w_n^{h,d,m,y} \right)}_{\text{cost to procure and sell energy}} \Bigg] \tag{6.3a}$$

which is to be minimized subject to the following constraints:

$$\text{s.t.} \quad \sum_{n,y} c_b^y b_n^y + c_s^y z_n^y \leq B, \tag{6.3b}$$

$$\sum_{a\in\mathcal{I}(n)} \tau_a(f_a^{h,d,m,y}) + \sum_{a\in\mathcal{O}(n)} \tau_a(-f_a^{h,d,m,y}) + R \cdot r_n^{h,d,m,y} + x_{O,n}^{h,d,m,y} + x_{N,n}^{h,d,m,y}$$

$$\geq d_n^{h,d,m,y} - w_n^{h,d,m,y} - v^{h,d} \left(\sum_{y'=1}^{y} \xi^{y-y'} z_n^{y'} \right), \tag{6.3c}$$

$$\sum_h w_n^{h,d,m,y} \leq \beta \sum_h \left[v^{h,d} \left(\sum_{y'=1}^{y} \xi^{y-y'} z_n^{y'} \right) + \max\left\{0, -d_n^{h,d,m,y}\right\} \right], \tag{6.3d}$$

$$s_n^{h+1,d,m,y} = \psi s_n^{h,d,m,y} - r_n^{h,d,m,y},$$

$$s_n^{1,d,m,y} = \psi s_n^{24,d,m,y} - r_n^{24,d,m,y},$$

$$s_n^{h,d,m,y} \leq \sum_{y'=1}^{y} \nu^{y-y'} b_n^{y'}, \tag{6.3e}$$

$$x_{O,n}^{h,d,m,y} \leq G_o^h, \; x_{N,n}^{h,d,m,y} \leq G_n^h, \tag{6.3f}$$

$$|f_a^{h,d,m,y}| \leq K_a, \tag{6.3g}$$

$$s_n^{h,d,m,y}, x_{O,n}^{h,d,m,y}, x_{N,n}^{h,d,m,y}, w_n^{h,d,m,y}, b_n^y, z_n^y \geq 0,$$

where $\tau_a(f) := \eta_a f_+ - f_- : f = f_+ - f_-, f_+, f_- \geq 0$ models the net flow through arc a after transmission losses, and all constraints are taken over the indices n, a, h, d, m, y

whenever these indices are present in a constraint and not prescribed by a sum.

Objective. Equation (6.3a) minimizes the time-discounted cost of OCP's investment plus the time-discounted cost of procuring electricity from the Moroccan national grid. Correspondingly, we minimize the sum of four terms which each correspond to annual strategic or operational decisions, and are discounted by year via a discount factor $\rho = 0.96$ for a discount rate of 4%.

The first term models the cost of installing batteries, minus their end-of-horizon salvage value. Similarly, the second term represents the cost of installing solar panels minus their salvage value. The third term corresponds to the operational cost of renting power lines. We take the planning horizon to be $T = 20$ years, and assume that batteries and solar panels degrade over time. Finally, the fourth term captures the operational cost of procuring energy from the Moroccan power grid (from either provider), minus the profit from selling energy to the grid.

Constraints. To ensure that the model is physically realistic, it contains a number of constraints. Constraint (6.3b) imposes that the total investment cannot exceed the allocated budget. Constraint (6.3c) imposes Kirchoff's current law at each node $n \in \mathcal{N}$, after accounting for transmission losses, batteries, and solar generation. Note that we treat power generation output at each factory as negative demand, and impose an inequality constraint to allow factories/solar panels to shed load, as this will rarely be profitable at optimality. Constraint (6.3d) imposes that, on a daily basis, we cannot sell more than a pre-specified fraction (here 20%) of the energy we locally produce at a given node (either via solar panels or via negative demand). This constraint is imposed both for legal purposes and to prevent energy providers from viewing OCP as a competitor in the energy generation business. Constraint (6.3e) ensures that, for each day, the amount of energy stored in batteries at hour $h + 1$ is equal to that at hour h, plus any extra energy injected into the battery, minus any injected into the grid, after accounting for efficiency losses. Moreover, the amount of energy stored in batteries at any one time cannot exceed capacity, or be negative. For the sake of

tractability, we assume that energy stored at hour 24 can be used at hour 1; this is a simplification, and assumes that each reduced scenario is succeeded by scenarios of the same type. Constraint (6.3f) ensures that the electricity procured from the grid cannot exceed its capacity. Finally, Constraint (6.3g) ensures that net flow through a line does not exceed the lines capacity.

6.2.3 Operationalizing the SAA Model

In this section, we convert the SAA model we derived in the previous section into one which OCP can use to make real-time decisions. This is an important practical issue. Indeed, the SAA model can be run at the beginning of each year of the planning horizon to make strategic decisions regarding how many solar panels and batteries OCP should install that year and where it should install them, but does not actually provide OCP with any guidance on how it should operate its system on an hourly basis. In particular, the SAA model proposes a daily policy for each reduced scenario in our set of scenarios, but, since the amount of sunlight is continuous and uncertain, real days will almost surely differ from these scenarios.

To resolve this issue, we now develop a real-time linear optimization model which, for a given time t in a given day d of year y, and a given—highly accurate—solar capacity forecast for the rest of the day, prescribes an optimal policy for how to operate OCP's system across the rest of the day. The model discretizes the first d days of year y into T time periods (in spirit, hours, although OCP could also develop a more granular model with more time periods), whereby $t \leq T$ is the current time period and T is the last time period of day d. Notice that, at time t, decision variables that correspond to times $i \leq t-1$ have been already decided (we use tilde to indicate such quantities). Fixing this data in our SAA model gives the following real-time model:

$$\begin{aligned}
\min \quad & \sum_{i=t}^{T} \left[\sum_{a} c^i_{r,a} |f^i_a| + \sum_{n} \left(p^i_{O,n} x^i_{O,n} + p^i_{N,n} x^i_{N,n} - p^i_{w,n} w^i_n \right) \right] \\
\text{s.t.} \quad & \sum_{a \in \mathcal{I}(n)} \tau_a(f^i_a) + \sum_{a \in \mathcal{O}(n)} \tau_a(-f^i_a) + x^i_{O,n} + x^i_{N,n} - w^i_n + R \cdot r^i_n
\end{aligned}$$

$$
\begin{aligned}
&\geq d_n^i - v_n^i \left(\sum_{y'=1}^{y} \xi^{y-y'} \tilde{z}_n^{y'} \right), && \forall n,\ t \leq i \leq T, \\
&\sum_{i=t}^{T} w_n^i \leq \beta \sum_{i=1}^{t-1} \left[v_n^i \left(\sum_{y'=1}^{y} \xi^{y-y'} \tilde{z}_n^{y'} \right) + \max\left\{0, -d_n^i\right\} \right] - \sum_{i=1}^{t-1} \tilde{w}_n^i, && \forall n, \\
&s_n^t = \psi \tilde{s}_n^{t-1} - \tilde{r}_n^{t-1},\ s_n^{i+1} = \psi s_n^{i-1} - r_n^{i-1}, && \forall n,\ t < i < T, \\
&s_n^{T+1} = \psi s_n^T - r_n^T, \\
&s_n^i \leq \sum_{y'=1}^{y} \nu^{y-y'} \tilde{b}_n^{y'}, && \forall n,\ t \leq i \leq T, \\
&x_{O,n}^i \leq G_o^i,\ x_{N,n}^i \leq G_n^i, && \forall n,\ t \leq i \leq T, \\
&|f_a^i| \leq K_a,\ s_n^i, x_{O,n}^i, x_{N,n}^i, w_n^i \geq 0, && \forall a,\ n,\ t \leq i \leq T,
\end{aligned}
\tag{6.4}
$$

where s_n^{T+1} is the amount of energy stored by OCP at the end of the day, which for simplicity we set to be equal to the amount of energy stored at the start of the day in order that the optimizer does not completely drain the batteries in time period T. Although we do not implement this here, we could also consider more complex approaches which address end-of-horizon effects more accurately, e.g., extending the planning horizon over multiple days or rewarding energy storage levels at the end of the day via a salvage function (see [120]). Problem (6.4) is a linear optimization problem with $O\big(T(|\mathcal{N}| + |\mathcal{A}|)\big)$ variables, and can thus be solved at scale in real-time.

OCP operates its system in time period t by acting according to an optimal value of the decision variables $(f_a^t, s_n^t, r_n^t, x_{O,n}^t, x_{N,n}^t, w_n^t)$ in Problem (6.4). Respectively, these variables correspond to the amount of power we transmit through each line, the amount of energy we store at each node's batteries, the amount of energy we charge or discharge from each node's batteries, the amount of power we procure from each provider at each node, and the amount of power we sell back to the grid in time period t. We remark that if the real-time solar capacity factors in time period $t+1$ deviate from their forecast value in time period t, then we can rerun the real-time model in period $t+1$ and proceed accordingly.

We close this section by noting that in the same way as solar forecasts deviates

over a period of days, their distribution deviates over a period of years, due to, e.g., climate change. To address this, OCP should also rerun the SAA model at the start of each year (with the number of solar panels and batteries previously purchased at each site fixed as sunk costs).

This approach is potentially useful for a second reason: the analysis in this chapter is built upon one year of hourly solar capacity data, since this is all OCP currently has access to. In an opposite direction, rerunning the SAA model at the start of each year allows OCP to iteratively incorporate more data in its forecasts, thus driving the implemented solutions closer to that we would obtain with full knowledge of the underlying stochastic process.

6.2.4 Solar Capacity Factor Data Description and Methodology

In this section, we outline the ML-driven scenario reduction methodology we invoke to reduce the number of days considered in each year of our planning horizon. The primary motivation for this approach is that optimizing over a small representative set of $|\mathcal{D}|$ days instead of all $365 \cdot |\mathcal{Y}| \cdot |\mathcal{N}|$ days in each year of the planning horizon reduce both the dimensionality of the problem and our policies sensitivity to outliers.

Simulated Solar Capacity Factors. We are given a year of simulated solar capacity factors for each OCP mining site, for a total of $365 \cdot |\mathcal{N}|$ simulated days. The simulated solar capacity factors are obtained using `PVsyst 7.2`, a software package for the study, sizing, and data analysis of complete PV systems. The simulations were performed by OCP engineers and were based on several years of historical solar generation data across all OCP sites.

We denote by N the total number of data points in our simulated solar capacity factor dataset, which we index by $i = 1, \ldots, N$. Each data point is 24-dimensional and represents the hourly solar capacity factor on a given day and at a given location. Correspondingly, the simulated solar capacity factor in hour $h \in \mathcal{H}$ of data point $i \in [N]$ is $v_0^{h,i}$.

Clustering-based Scenario Reduction. We obtain a reduced set of scenarios $\mathcal{D}$ by clustering simulated solar capacity factors. We apply the standard K-means clustering method, which minimizes the within-cluster variances using a coordinate descent algorithm. The resulting cluster assignment provides us with a function $c(\cdot) : [N] \mapsto \mathcal{D}$, which maps simulated days to their assigned reduced scenarios. Formally, we perform clustering by minimizing:

$$\min_{c(\cdot),(\bar{\boldsymbol{v}}^d)_{d\in\mathcal{D}}} \sum_{d\in\mathcal{D}} \sum_{i:c(i)=d} \|\boldsymbol{v}_{\mathbf{0}}^i - \bar{\boldsymbol{v}}^d\|_2^2.$$

The centroids of the estimated clusters are taken as the reduced scenarios. In particular, the solar capacity factor for hour $h \in \mathcal{H}$ of reduced scenario $d \in \mathcal{D}$ is calculated as $\bar{v}^{h,d} = \frac{\sum_{i:c(i)=d} v_0^{h,i}}{|\{i:c(i)=d\}|}$. As discussed previously, each scenario $d \in \mathcal{D}$ occurs with probability $P^{d,m,y}$ in month $m \in \mathcal{M}$ in year $y \in \mathcal{Y}$, which is empirically estimated as the number of days from month m that belong in the cluster of reduced scenario d in our simulated dataset. In Section 6.3, we explore techniques for robustifying these point estimates to account for uncertainty in renewable generation output.

6.3 Immunizing Against Uncertainty in Solar Generation

When we introduced our SAA model in Section 6.2, we assumed that the amount of sunlight in a given day is drawn from a probability distribution at the start of the day, and subsequently deterministic. Unfortunately, this is not true in practice: when OCP plans its operations for a given day, it has access to a highly accurate forecast of the amount of sunlight available, but this forecast is not deterministically accurate. To address this, OCP solves a real-time problem in each hour of the day, as suggested in Section 6.2.3. However, this strategy is suboptimal for two reasons. First, the SAA problem does not account for deviations between the SAA and real-time solutions. Second, the real-time problem in each time period t does not account for further deviations if the weather forecast changes in period $t + 1$.

From a RO perspective, the SAA model also suffers from a further defect: the amount of solar generation in each day of the planning horizon is drawn from an underlying probability distribution, while we have access to a sample from this distribution but not the distribution itself. Thus, the SAA model overfits the simulated data in a finite-sample setting and underperforms out-of-sample.

Accordingly, in this section, we robustify the model in two ways. First, we address the deviation between the SAA and real-time problems: we outline how we model uncertainty in solar generation in Section 6.3.1, perform an exploratory analysis of OCP's solar capacity factor data in Section 6.3.2, use our analysis to propose a data-driven uncertainty set in Section 6.3.3, and robustify the SAA model against deviations from forecasts in Section 6.3.4. Second, we immunize the model against overfitting to the simulated data by using recent advances in DRO in Section 6.3.5. Interestingly, this robustification does not impact the SAA problem's tractability, since the robust model can be reformulated as a tractable linear optimization model with exponential cone constraints, featuring a number of decision variables and constraints proportional to the SAA model. All in all, our approach immunizes the SAA model against uncertainty in a tractable and unified fashion.

6.3.1 Uncertainty in Solar Generation

In the reduced SAA formulation (6.3) laid out in the previous section, we assumed that the solar capacity factor $v^{h,d}$ for hour $h \in \mathcal{H}$ of scenario $d \in \mathcal{D}$ is unknown at the start of each year of the planning horizon, and takes the value $v^{h,d} := \bar{v}^{h,d}$ before OCP plans its operations in scenario $\mathcal{D}$. This model of uncertainty is analogous to OCP being given a probability distribution of capacity factors when making its asset purchases at the start of each year, and subsequently receiving a perfectly accurate forecast at the beginning of each day d, for the entirety of day d, which allows OCP to plan its operations for that day without accounting for uncertainty.

In practice, the true solar capacity factors at each node n and time (h, d, m, y) are highly correlated to the forecasts OCP receives at time $(1, d, m, y)$, but not exactly equal to them. To account for this discrepancy, we now introduce a vector of uncertain

parameters $\boldsymbol{u}$ and require that our OCP's operational policy is feasible for all capacity factors $\boldsymbol{v}$ in the set

$$\mathcal{U} = \left\{ \boldsymbol{v} \in \mathbb{R}_{+}^{|\mathcal{H}|\times|\mathcal{D}|\times|\mathcal{M}|\times|\mathcal{Y}|\times|\mathcal{N}|} : \quad v_n^{h,d,m,y} = \bar{v}^{h,d} + u_n^{h,d,m,y} \quad \forall h, d, m, y, n \quad \forall \boldsymbol{u} \in \mathcal{U}_u \right\}. \tag{6.5}$$

In the rest of this section, we describe how we model the uncertainty in solar generation through a tailored, data-driven uncertainty set $\mathcal{U}_u$, which accounts for adversarial perturbations in the nominal solar generation, albeit in a controlled and structured way.

6.3.2 Exploring the Uncertainty

We now study the predictability of the solar capacity factors to, in the next section, design a data-driven uncertainty set that accurately reflects uncertainty in renewable generation output without being excessively conservative. Correspondingly, the main quantity which we are interested in is the within-reduced-scenario distance between an average hourly solar capacity factor and a given simulated capacity factor. Therefore, we explore how this quantity evolves across time in order to propose an uncertainty set which accurately models deviations in solar generation capacity. For the analysis that follows, we take $|\mathcal{D}| = 10$.

We first study two common reduced scenarios obtained by clustering our solar capacity factors according to the methodology described in Section 6.2.4. As shown in Figure 6-2 (left), the first reduced scenario (RS1) is a high-generation scenario that occurs throughout the year, while the second reduced scenario (RS2) is a low-generation scenario that occurs mostly in spring and fall months. Figure 6-2 (right) presents, for both RS1 and RS2, the mean and standard deviation in the hourly solar generation capacity factors. The mean corresponds to the nominal solar generation $\bar{v}^{h,d}$, while the standard deviation varies depending on the hour h of the day and is typically zero during nighttime. Unsurprisingly, the mean solar generation for RS1 is higher. More interestingly however, the variance for RS2 is higher than RS1. This is perhaps best explained by the more variable cloudy weather conditions that RS2

captures, and suggests that the structure of uncertainty varies between scenarios and should be modeled as such.

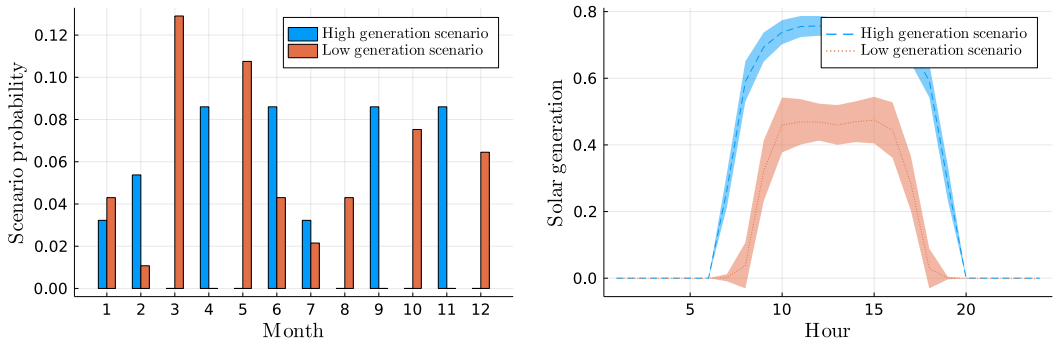


Figure 6-2: Probability of occurrence during each month of selected reduced scenarios (left) and mean and standard deviation of solar generation for selected reduced scenarios (right).

Figure 6-3 presents the empirical distributions of the maximum absolute uncertainty during the entire day for RS1 and RS2 (Figure 6-3 (left)), $\max_h |u_n^{h,d,m,y}|$, and the hour at which the maximum absolute uncertainty occurs (Figure 6-3 (right)), $\arg\max_h |u_n^{h,d,m,y}|$. The maximum absolute uncertainty is a small fraction of the corresponding nominal value, is notably higher for the low-generation RS2, and occurs during sunrise or sunset.

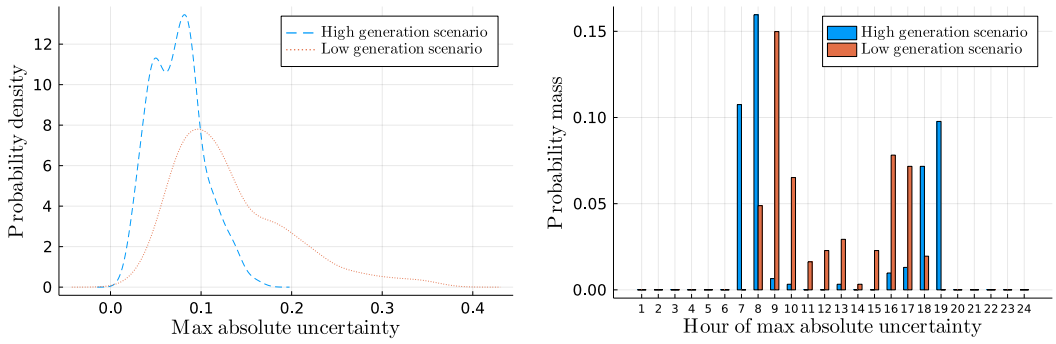
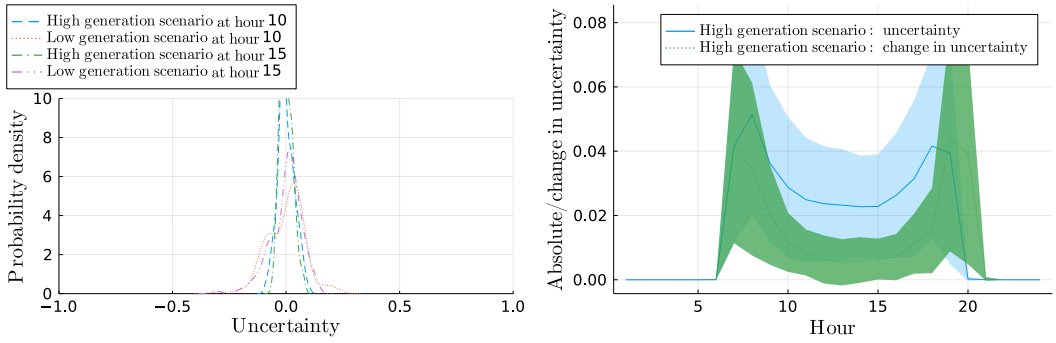


Figure 6-3: Insights from the empirical distribution of the uncertainty. The maximum absolute uncertainty is small for high generation scenarios, but larger for low generation ones (left). Moreover, the maximum uncertainty is observed during sunset or sunrise for RS1, and throughout the day for RS2 (right).

Figure 6-4 investigates the empirical distribution of deviations between solar capacity factors and their forecasts, and the change in uncertainty between consecutive hours. Figure 6-4 (left) investigates the empirical distribution of uncertainty, $u_n^{h,d,m,y}$: across all reduced scenarios, the distribution is centered and concentrated around zero. Figure 6-4 (right) shows the mean and standard deviation of the absolute uncertainty, $|u_n^{h,d,m,y}|$ and the absolute change in uncertainty during consecutive hours, $|u_n^{h,d,m,y} - u_n^{h-1,d,m,y}|$. We observe that both quantities are small compared to the nominal solar generation values, and peak during sunrise and sunset. Further, the absolute change in uncertainty is, on average, smaller than the absolute uncertainty. This observation suggests that, if a realized day is above the mean solar generation of its corresponding reduced scenario at hour $h-1$, it will likely also be at hour h.



(a) (b)

Figure 6-4: The empirical distribution of the uncertainty per hour and of the daily sum are both centered and concentrated around zero (a). Moreover, mean absolute uncertainty/change in uncertainty during consecutive hours is small and is maximized during sunrise/sunset (b). All in all, mean uncertainty/change in uncertainty is only a small fraction of mean solar generation.

We now summarize the main managerial insights from our analysis and their implications:

- Figures 6-2-6-3 suggest that uncertain parameters during different hours of the day, as well as during different reduced scenarios, should be modeled separately in order to obtain as accurate a model as possible. However, provided that the number of reduced scenarios is sufficiently large, we need not treat uncertain parameters differently depending on which month they correspond to.

- Figure 6-4 (left) suggests uncertainty sets motivated by the CLT may perform well in practice.
- Figure 6-4 (right) implies that "smoothness constraints" which link uncertainty during consecutive hours and require that it varies slowly, may perform well in practice.

Motivated by these findings, we propose a data-driven uncertainty set in the next subsection.

6.3.3 A Data-Driven Uncertainty Set

We now propose a data-driven uncertainty set which immunizes the sample-robust model proposed in the previous section against intermittency in solar generation capacity. This immunization is significant, since the SAA model we derived in the previous section relies on day-ahead solar capacity forecasts being deterministically accurate, which does not occur in practice.

As discussed in Section 6.3.1, the (uncertain) solar generation at node n and time (h, d, m, y) is taken to be $v_n^{h,d,m,y} = \bar{v}^{h,d} + u_n^{h,d,m,y}$, where $u_n^{h,d,m,y}$ is a component of the vector of uncertain parameters $\boldsymbol{u} \in \mathcal{U}_u$. Moreover, as discussed in Section 6.3.2, the practical behaviour of $\mathcal{U}_u$ can be characterized by three constraints. First, the absolute value of $u_n^{h,d,m,y}$ is never too large in any hour. Second, $v_n^{h,d,m,y}$ never changes by too much between an hour and the previous one. Third, $v_n^{h,d,m,y}$ obeys the Central Limit Theorem in aggregate. Combining the ideas from Sections 6.3.1 and 6.3.2 therefore gives rise to the following uncertainty set:

$$\begin{aligned}\mathcal{U}_u = \{ \boldsymbol{u} \in \mathbb{R}^{|\mathcal{H}|\times|\mathcal{D}|\times|\mathcal{M}|\times|\mathcal{Y}|\times|\mathcal{N}|} : \quad & |u_n^{h,d,m,y}| \leq U_{\max}^{h,d}, \\ & \left|u_n^{h,d,m,y} - u_n^{h-1,d,m,y}\right| \leq U_{\mathrm{C}}^{h,d} \quad \forall h, d, m, y, n, , \left| \sum_{n,h,m,y} u_n^{h,d,m,y} \right| \leq U_{\mathrm{CLT}}^{d} \quad \forall d \Bigg\}.\end{aligned} \tag{6.6}$$

This set is parameterized by the following three sets of parameters, which we now define and provide guidance on how to set their values:

- $U^{h,d}_{\max} \in \mathbb{R}_+$: The maximum allowable perturbation for hour h of reduced scenario d. We estimate $U^{h,d}_{\max}$ as a fraction $\gamma_{\max} \geq 0$ of the maximum absolute deviation from the cluster centroid across all data points that are assigned to the same cluster: $U^{h,d}_{\max} = \gamma_{\max} \cdot \max_{i:c(i)=d} \left| v_0^{h,i} - \bar{v}^{h,d} \right|$.

- $U^{h,d}_{\mathrm{C}} \in \mathbb{R}_+$: Imposes smoothness across time, i.e., the uncertainty in solar generation cannot be too different between hours $h-1$ and h of reduced scenario d. We estimate $U^{h,d}_{\mathrm{C}}$ as a fraction $\gamma_{\mathrm{C}} \geq 0$ of the maximum absolute difference in uncertainty across all data points that are assigned to the same cluster: $U^{h,d}_{\mathrm{C}} = \gamma_{\mathrm{C}} \cdot \max_{i:c(i)=d} \left| \left(v_0^{h,i} - \bar{v}^{h,d} \right) - \left(v_0^{h-1,i} - \bar{v}^{h-1,d} \right) \right|$.

- $U^{d}_{\mathrm{CLT}} \in \mathbb{R}_+$: Controls the limiting behavior of the realized uncertainty. For each reduced scenario d, and for fixed n, m, y, we focus on the aggregated daily uncertainty $\sum_h u_n^{h,d,m,y}$. The central limit theorem implies that the sum across n, m, y of the realized aggregated daily uncertainties $\sum_{n,m,y} \left(\sum_h u_n^{h,d,m,y} \right)$ converges to a zero-mean normal distribution with standard deviation σ^d. We take σ^d as the corrected sample standard deviation and estimate U^{d}_{CLT} as a fraction $\gamma_{\mathrm{CLT}} \geq 0$ of the sample standard deviation: $U^{d}_{\mathrm{CLT}} = \gamma_{\mathrm{CLT}} \cdot \sigma^d$.

6.3.4 A Constraint-Aggregation Approach

In this section, we integrate the uncertainty set derived in section 6.3.3 within our SAA model. This is non-trivial: in our SAA model the uncertain solar capacities $v_n^{h,d,m,y}$ appears in constraints (6.3c) and (6.3d) individually. Therefore, as is well documented in the RO literature (see, e.g., [194]), directly imposing robustness constraint-wise is equivalent to simultaneously setting the capacity factors $v_n^{h,d,m,y}$ to their minimum value in each hour of the planning horizon and at each node in the network, while completely ignoring the smoothness and CLT constraints. To avoid this, we adopt a constraint aggregation approach which requires that the aggregated demand across all nodes and all time periods is feasible for all realizations of the uncertain parameter, and the total amount of energy sold across all nodes and all time periods satisfies legal requirements.

To simplify our notation for the rest of this section, we now concatenate all variables except the ones corresponding to solar investment decisions into the vector:

$$\boldsymbol{x} := (\boldsymbol{f}, \boldsymbol{r}, \boldsymbol{x}_O, \boldsymbol{x}_N, \boldsymbol{w}).$$

Further, we denote the aggregated installed solar capacity in node n and in year y by

$$\bar{z}_n^y := \sum_{y'=1}^{y} \xi^{y-y'} z_n^{y'}.$$

By doing so, in both constraints of interest, the coefficients of $\boldsymbol{x}$ are deterministic, whereas the coefficients of $\bar{\boldsymbol{z}}$ involve the uncertain parameters. Let us first focus on the demand constraints (6.3c). The resulting aggregated constraint, which needs to hold for all realizations of $\boldsymbol{v} \in \mathcal{U}$, is

$$\begin{aligned}\sum_{n,h,d,m,y} \Bigg[&\sum_{a \in \mathcal{I}(n)} \tau_a(f_a^{h,d,m,y}) + \sum_{a \in \mathcal{O}(n)} \tau_a(-f_a^{h,d,m,y}) + v^{h,d} \left(\sum_{y'=1}^{y} \xi^{y-y'} z_n^{y'} \right) \\ &+ R \cdot r_n^{h,d,m,y} + x_{O,n}^{h,d,m,y} + x_{N,n}^{h,d,m,y} - w_n^{h,d,m,y} \Bigg] \geq \sum_{n,h,d,m,y} d_n^{h,d,m,y}, \qquad \forall \boldsymbol{v} \in \mathcal{U}. \end{aligned} \tag{6.7}$$

Moreover, it is not too hard to see that this condition holds if and only if

$$\boldsymbol{a_1}^\top \boldsymbol{x} + \min_{\boldsymbol{v} \in \mathcal{U}} \sum_{n,h,d,m,y} v_n^{h,d,m,y} \bar{z}_n^y \geq b_1,$$

where we denote by $\boldsymbol{a_1}$ all deterministic coefficients in Equation (6.3c) (i.e., all coefficients except $\boldsymbol{v}$) and by $b_1 := \sum_{n,h,d,m,y} d_n^{h,d,m,y}$. We now derive this constraint's deterministic equivalent via the following lemma, which holds due to strong duality (proof deferred to Section 6.6):

Lemma 13. *The robust constraint* (6.7) *admits the following deterministic reformu-*

lation:

$$
\begin{aligned}
&\boldsymbol{a_1}^\top \boldsymbol{x} + \left[\sum_d \left(\sum_{n,h,m,y} \bar{v}^{h,d} \sigma_{1,n}^{h,d,m,y} + U_{\max}^{h,d} \phi_{1,n}^{h,d,m,y} + U_{\max}^{h,d} \chi_{1,n}^{h,d,m,y} + U_C^{h,d} \psi_{1,n}^{h,d,m,y} + U_C^{h,d} \omega_{1,n}^{h,d,m,y} \right) \right. \\
&\quad \left. + U_{CLT}^d \tau_1^d \right] \geq b_1, \\
&\sigma_{1,n}^{h,d,m,y} \leq \bar{z}_n^y, \quad \forall h,d,m,y,n, \\
&-\sigma_{1,n}^{h,d,m,y} - \phi_{1,n}^{h,d,m,y} + \chi_{1,n}^{h,d,m,y} - \psi_{1,n}^{h,d,m,y} + \omega_{1,n}^{h,d,m,y} + \psi_{1,n}^{h+1,d,m,y} - \omega_{1,n}^{h+1,d,m,y} + \tau_1^d = 0, \quad \forall h,d,m,y,n, \\
&\phi_{1,n}^{h,d,m,y} \leq 0,\ \chi_{1,n}^{h,d,m,y} \leq 0,\ \psi_{1,n}^{h,d,m,y} \leq 0,\ \omega_{1,n}^{h,d,m,y} \leq 0,\ \tau_1^d \leq 0, \quad \forall h,d,m,y,n.
\end{aligned}
\tag{6.8}
$$

We now focus on constraints (6.3d). In this case, the resulting aggregated constraint, which needs to hold for all realizations of $\boldsymbol{v} \in \mathcal{U}$, can be written as

$$
\begin{aligned}
&\sum_{n,y} \left[\sum_{h,d,m} w_n^{h,d,m,y} - \beta v_n^{h,d,m,y} \bar{z}_n^y \right] \leq \beta \sum_{n,h,d,m,y} \max\left\{0, -d_n^{h,d,m,y}\right\}, \qquad \forall \boldsymbol{v} \in \mathcal{U}, \\
&\Leftrightarrow \quad \boldsymbol{a_2}^\top \boldsymbol{x} + \max_{\boldsymbol{v} \in \mathcal{U}} \sum_{n,h,d,m,y} \beta v_n^{h,d,m,y} \bar{z}_n^y \leq b_2,
\end{aligned}
\tag{6.9}
$$

where we denote by $\boldsymbol{a_2}$ all deterministic coefficients in Equation (6.3d) (i.e., all coefficients except the ones that involve $\boldsymbol{v}$) and by $b_2 := \beta \sum_{n,h,d,m,y} \max\left\{0, -d_n^{h,d,m,y}\right\}$. Applying the same machinery as we did in Lemma 13 and denoting by $\boldsymbol{\sigma_2}, \boldsymbol{\tau_2}, \boldsymbol{\phi_2}, \boldsymbol{\chi_2}, \boldsymbol{\psi_2}, \boldsymbol{\omega_2}$ the corresponding dual variables, yields the deterministic reformulation:

$$
\begin{aligned}
&\boldsymbol{a_2}^\top \boldsymbol{x} + \left[\sum_d \left(\sum_{n,h,m,y} \bar{v}^{h,d} \sigma_{2,n}^{h,d,m,y} + U_{\max}^{h,d} \phi_{2,n}^{h,d,m,y} + U_{\max}^{h,d} \chi_{2,n}^{h,d,m,y} + U_{\mathrm{C}}^{h,d} \psi_{2,n}^{h,d,m,y} + U_{\mathrm{C}}^{h,d} \omega_{2,n}^{h,d,m,y} \right) \right. \\
&\quad \left. + U_{\mathrm{CLT}}^d \tau_2^d \right] \leq b_2, \\
&\sigma_{2,n}^{h,d,m,y} \leq \beta \bar{z}_n^y, \quad \forall h,d,m,y,n, \\
&-\sigma_{2,n}^{h,d,m,y} - \phi_{2,n}^{h,d,m,y} + \chi_{2,n}^{h,d,m,y} - \psi_{2,n}^{h,d,m,y} + \omega_{2,n}^{h,d,m,y} + \psi_{2,n}^{h+1,d,m,y} - \omega_{2,n}^{h+1,d,m,y} + \tau_2^d = 0, \quad \forall h,d,m,y,n, \\
&\phi_{2,n}^{h,d,m,y} \leq 0,\ \chi_{2,n}^{h,d,m,y} \leq 0,\ \psi_{2,n}^{h,d,m,y} \leq 0,\ \omega_{2,n}^{h,d,m,y} \leq 0,\ \tau_2^d \leq 0, \quad \forall h,d,m,y,n.
\end{aligned}
\tag{6.10}
$$

Finally, we obtain our RO model by imposing constraints (6.8) and (6.10) in the SAA model. The resulting problem is notably a linear problem with $2 \cdot |\mathcal{D}| \cdot (5 \cdot |\mathcal{N}| \cdot |\mathcal{H}| \cdot |\mathcal{M}| \cdot |\mathcal{Y}| + 1)$ new variables and $2 \cdot |\mathcal{N}| \cdot |\mathcal{H}| \cdot |\mathcal{D}| \cdot |\mathcal{M}| \cdot |\mathcal{Y}| + 2$ new constraints, and can thus be tractably solved at scale.

6.3.5 Preventing Overfitting: Distributionally Robust Optimization to the Rescue

In this section, we immunize our model against overfitting the solar capacity factors provided by OCP, using ideas from DRO. This is an important practical step. Indeed, as mentioned in the introduction, sample average approximations of capacity expansion problems generally perform well in large sample settings, but overfit in the presence of small sample sizes. Moreover, Moroccan weather patterns may change due to changes in the climate induced by increasing levels of carbon in the atmosphere, making historical data unreliable. Therefore, overfitting could certainly occur in our problem setting, where we have access to capacity factors on an hourly basis for one year.

To prevent this, we take a DRO approach to SAA very much inspired by the works of [214, 16] among others. Specifically, for ease of notation, let us vectorize the strategic decisions (investment in batteries and solar panels across all sites and years) as $\boldsymbol{z}_s := (\boldsymbol{b}, \boldsymbol{z})$ and the operational decisions (cost to rent lines, cost to procure and sell energy) as $\boldsymbol{x} := (\boldsymbol{f}, \boldsymbol{x}_O, \boldsymbol{x}_N, \boldsymbol{w})$ that appear in the objective. In addition, we denote by $\boldsymbol{c}_z$ and $\boldsymbol{c}_x$ the corresponding vectors of objective coefficients for the investment decisions (e.g., time-discounted cost of purchasing batteries) and operational decisions (e.g., time-discounted cost of renting lines), respectively. We then rewrite our model as:

$$\min_{(\boldsymbol{z}_s, \boldsymbol{x}) \in \mathcal{Z}} \quad \underbrace{\boldsymbol{c}_z^\top \boldsymbol{z}_s}_{\text{strategic decisions}} + \underbrace{\sum_{y,m,d} D^{m,y} P^{d,m,y} (\boldsymbol{c}_x^{m,y})^\top \boldsymbol{x}^{d,m,y}}_{\text{operational decisions}}, \tag{6.11}$$

where $\mathcal{Z}$ denotes the feasible set defined by constraints (6.3b)-(6.3g), (6.8) and (6.10).

We now improve this model, by optimizing over the worst case probability measure $\boldsymbol{Q}$ within a KL-divergence of δ from the empirical measure $\boldsymbol{P}$; recall that the KL-divergence of two measures $\boldsymbol{Q}$, $\boldsymbol{P}$ defined on the same probability space is the asymmetric distance

$$\mathrm{D}_{\mathrm{KL}}(\boldsymbol{Q} || \boldsymbol{P}) := \sum_{d \in \mathcal{D}} Q^d \log\left(\frac{Q^d}{P^d}\right).$$

As shown by [214], this approach behaves optimally in terms of minimizing the out-of-sample disappointment over all variants of SAA. Moreover, as shown by [16], this approach outperforms SAA in the small sample size regime. Formally, we have the DRO problem:

$$\min_{(\boldsymbol{z}_s,\boldsymbol{x})\in\mathcal{Z}} \quad \boldsymbol{c}_z^\top \boldsymbol{z}_s + \sum_{y,m} D^{m,y} \max_{\boldsymbol{Q}^{m,y}\in\mathcal{Q}(\boldsymbol{P}^{m,y},\delta)} \sum_d Q^{d,m,y} (\boldsymbol{c}_x^{m,y})^\top \boldsymbol{x}^{d,m,y} \tag{6.12}$$

where $\mathcal{Q}(\boldsymbol{P},\delta) := \{\boldsymbol{Q}\in\mathbb{R}_+^{|\mathcal{D}|} : \boldsymbol{e}^\top\boldsymbol{Q}=1, \sum_{d\in\mathcal{D}} Q^d \log(Q^d/P^d)\leq\delta\}$ denotes the set of all probability measures within a KL-divergence of δ of the empirical one. We now reformulate the DRO problem as a deterministic one, via the following lemma which leverages strong conic duality and is essentially due to [147] (Theorem 1):

Lemma 14. *Problem* (6.12) *admits the following deterministic reformulation:*

$$\begin{aligned}\min_{\substack{(\boldsymbol{z}_s,\boldsymbol{x})\in\mathcal{Z},\\ \boldsymbol{\alpha}\in\mathbb{R}^{|\mathcal{M}|\cdot|\mathcal{Y}|},\\ \boldsymbol{\beta}\in\mathbb{R}^{|\mathcal{M}|\cdot|\mathcal{Y}|}\geq 0,\\ \boldsymbol{\gamma},\boldsymbol{\zeta}\in\mathbb{R}^{|\mathcal{D}|\cdot|\mathcal{M}|\cdot|\mathcal{Y}|}}} & \boldsymbol{c}_z^\top\boldsymbol{z}_s + \sum_{y,m}\left(D^{m,y}\alpha^{m,y} + \delta\beta^{m,y} + \sum_d P^{d,m,y}\gamma^{d,m,y}\right)\\ \text{s.t.} \quad & \alpha^{m,y} - \zeta^{d,m,y} \geq (\boldsymbol{c}_x^{m,y})^\top \boldsymbol{x}^{d,m,y},\ (-\beta^{d,m},\zeta^{d,m,y},\gamma^{d,m,y})\in\mathcal{K}_{exp}^\star, \end{aligned} \tag{6.13}$$

$$\textit{where:} \quad \mathcal{K}_{exp}^\star := \mathrm{cl}\left(\left\{(u,v,w)\in\mathbb{R}^3 : -u\exp(v/u)\leq\exp(1)w, u<0\right\}\right)$$

denotes the dual cone to the exponential cone and cl *denotes the closure of a set (see [196] for a derivation of the cone).*

We remark that we could replace the KL divergence with another ϕ-divergence or distance measure without significantly impacting the tractability of the deterministic reformulation; see [190] for a review of other DRO formulations. Lemma 14 reveals that we can convert our SAA problem into a DRO one while only increasing the number of decision variables by $2\cdot|\mathcal{M}|\cdot|\mathcal{Y}| + 2\cdot|\mathcal{D}|\cdot|\mathcal{M}|\cdot|\mathcal{Y}|$ and the number of (dual) exponential cone constraints by $|\mathcal{D}|\cdot|\mathcal{M}|\cdot|\mathcal{Y}|$. Thus, our DRO formulation can be tractably solved by state-of-the-art conic solvers such as `Mosek`.

6.4 Experiments

In this section, we describe how the deterministic and robust optimization methodologies proposed in Sections 6.2—6.3 are implemented in practice. We first detail a cross-validation procedure which allows the hyperparameters in the robust optimization models described in Section 6.3 to be set optimally (Section 6.7), study the cross-validated robust model's sensitivity to noise, and establish its stability under small perturbations in the data. Next, we explore the relationship between OCP's investment level and long-run operational costs and carbon emissions (Section 6.4.2), and establish that investing 10 billion MAD (resp. 20 billion MAD) reduces OCP's carbon emissions by 60% (resp. 80%) in a profitable fashion. Finally, we study the investment policy prescribed by the model with a 10 billion MAD budget in detail (Section 6.4.3), and demonstrate that the model both anticipates solar generation before it occurs and captures notions of load shifting.

Computing Resources. We implement our approach in `Julia` version 1.5 using `JuMP.jl` version 0.21. We solve the models using the `Mosek` commercial solver (version 9.2). All experiments were performed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running CentOS release 7.

6.4.1 Cross-Validation and Sensitivity Analysis

In this section, we propose a methodology for cross-validating the RO and DRO hyperparameters, and demonstrate that properly selecting these parameters leads to cost savings for OCP both in and out of sample. We also investigate the cross-validated model's sensitivity to distributional shifts between the training data and the testing data, e.g., induced by climate change.

Cross-Validation Methodology. Following a standard machine learning paradigm, we randomly split the year of site-wise solar generation data supplied by OCP into training, validation, testing sets on a month-by-month basis (with a training/validation/test split of 20 days/4 days/4 days). Using the training data, we estimate reduced scenarios

using the methodology laid out in Section 6.2.4, and solve Problem (6.13) to optimality for each value of the hyperparameters

$$(\gamma_{\max}, \gamma_C, \gamma_{\text{CLT}}, \delta) \in \left\{0, \frac{1}{2}, 1\right\}^3 \times \{0, 0.001, 0.01, 0.1, 1.0\}\,.$$

To select the optimal hyperparameters, we evaluate the performance of each model on the holdout validation set, and select the combination of parameters which performs best. To reduce the dimensionality of the problem, we use an investment budget of 10Bn MAD and set the number of reduced scenarios to 10. Moreover, to reduce the sensitivity of our approach to noise, we repeat the entire validation process five times, and take the hyperparameters that perform best on average.

The Impact of Robustness. Figure 6-5 depicts the average relative improvement in operational costs predicted by the validation set, compared to the SAA model, for a selected set of values of the RO and DRO hyperparameters. We observe that the RO- and DRO-guarded models provide an improvement of 6% over the SAA model; nonetheless, being overly conservative harms the model's performance (as can be seen from the top-right block of Figure 6-5). On the validation dataset, the combination of hyperparameters that performs best is $(\gamma_{\max}, \gamma_C, \gamma_{\text{CLT}}, \delta) = (0.5, 0.5, 1.0, 0.001)$ (instance-wise details for all examined combinations deferred to Section 6.7).

Effect of Robustness on the Investment. Next, we study the effect of introducing robustness in the model on the amount invested in solar panels (Figure 6-6 (left)) and in batteries (Figure 6-6 (right)), with a constant investment budget of 10 billion MAD. We observe that increasing the amount of uncertainty in the problem by increasing the RO uncertainty and/or DRO ambiguity set size results in a larger investment in solar panels and a smaller investment in batteries. This is perhaps surprising, since batteries offer OCP a recourse action after uncertainty in the system is revealed, while solar panels do not. However, it can be explained by three substructures in OCP's problem. First, roughly speaking, batteries allow load shifting to take place and reduce the overall cost accrued in hours of the day where the sun does not shine, while solar panels

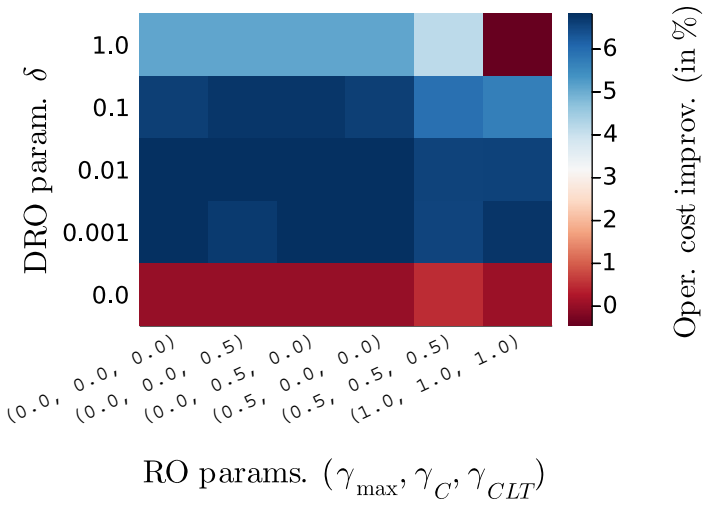
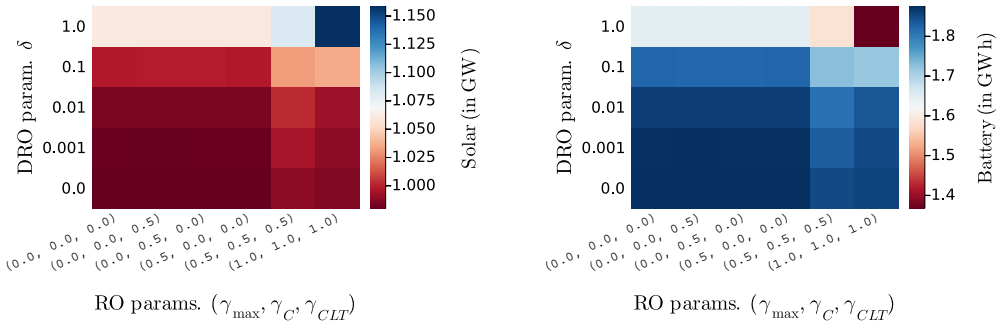


Figure 6-5: Cross-validating the RO and DRO budgets improves operational costs.

reduce the cost accrued in hours where the sun does shine. Second, in OCP's problem, the marginal price of electricity is highest in peak periods where the sun does shine. Third, due to the nature of the uncertainty sets designed in Section 6.3, increasing the size of the RO uncertainty and DRO ambiguity sets allows nature to increase the relative cost of the most expensive hours of the day. Correspondingly, when the overall investment budget is held constant, increasing the amount of robustness increases the amount invested in solar panels and decreases the amount invested in batteries.



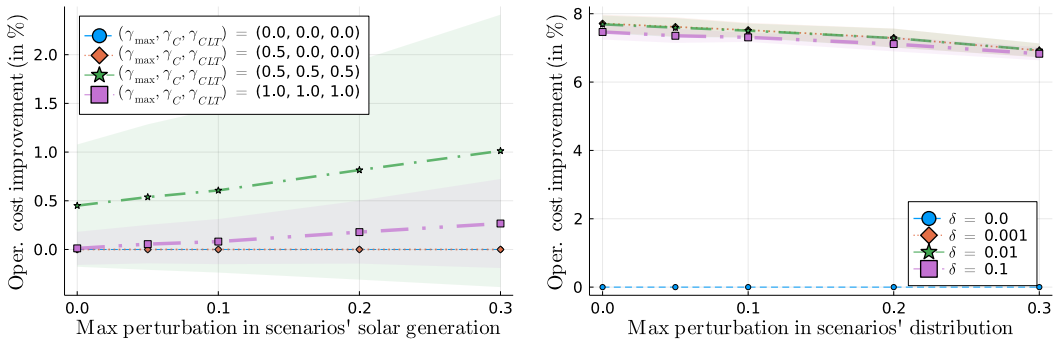
(a) (b)

Figure 6-6: Effect of RO and DRO uncertainty budgets on investment in solar panels and batteries with a 10 billion MAD (approx. 1 Bn USD) investment budget. Increasing robustness increases investment in solar panels (left) and decreases investment in batteries (right).

Sensitivity to Weather and Climate Shifts. We next investigate the model's sensitivity to two different types of perturbation in the solar generation data. To model variability in daily solar capacities (uncertainty in the weather), we generate synthetic test sets by perturbing the hourly solar capacity factors of each reduced scenario by an amount uniformly drawn from $[-\theta_1 \cdot \max(\overline{\boldsymbol{v}}), \theta_1 \cdot \max(\overline{\boldsymbol{v}})]$, where $\theta_1 \cdot \max(\overline{\boldsymbol{v}})$ corresponds to the maximum allowable perturbation as a fraction of the maximum solar capacity factor in the data. We examine $\theta_1 \in \{0, 0.05, 0.1, 0.2, 0.3\}$ and, for simplicity in presentation, we only focus on a subset of combinations of the RO hyperparameters $(\gamma_{\max}, \gamma_C, \gamma_{\text{CLT}})$. To assess the marginal effect of RO, we set the DRO hyperparameter $\delta = 0$. We independently generate 5 perturbed datasets for each value of θ_1 and report the mean and standard deviation of the improvement in operational costs compared to the SAA model. As shown in Figure 6-7 (left), RO effectively protects the model against various levels of perturbation of this type, providing an improvement of $0.5 - 1\%$ in operational costs over the SAA model.

To capture variability in seasonal weather patterns induced by climate change (uncertainty in the climate), we generate synthetic test sets by perturbing the reduced scenarios' monthly distributions in each year by randomly chosen amounts in $[-\theta_2, \theta_2]$, and properly re-scaling the resulting perturbed distributions. We examine $\theta_2 \in \{0, 0.05, 0.1, 0.2, 0.3\}$ and different values of the DRO hyperparameter δ in Figure 6-7 (right). To assess the marginal effect of DRO, we set all RO hyperparameters to 0. We independently generate 5 perturbed datasets for each value of θ_2 and report the mean and standard deviation of the improvement in operational costs compared to the SAA model. For perturbations of this type with a relative magnitude between 5 and 30%, DRO improves OCP's overall cost of operations by as much as 8% compared to the SAA solution.

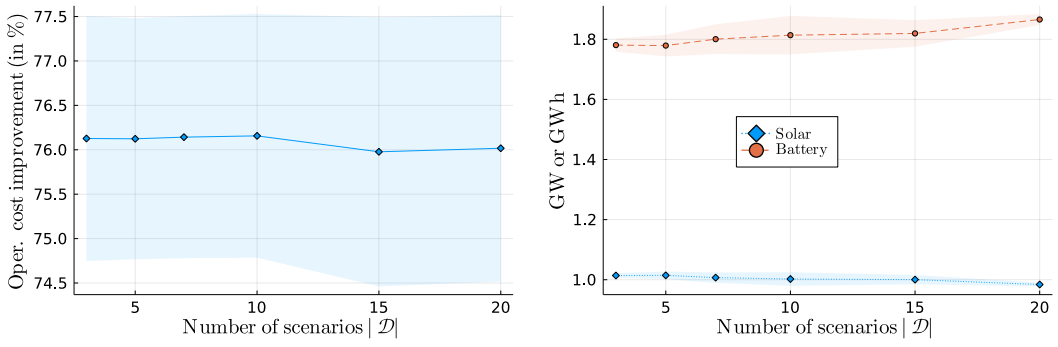
Sensitivity to the Number of Scenarios. We now explore the model's sensitivity to the number of reduced scenarios. We use an investment budget of 10Bn MAD, set the RO and DRO hyperparameters to $(\gamma_{\max}, \gamma_C, \gamma_{\text{CLT}}, \delta) = (0.5, 0.5, 0.5, 0.01)$ as suggested by our cross-validation procedure and sensitivity analysis, and vary the



(a) (b)

Figure 6-7: Performance of RO and DRO using perturbed solar generation data. We observe that RO protects against weather shifts (left) and DRO effectively guards against climate shifts (right).

number of scenarios in $\{3, 5, 7, 10, 15, 20\}$. Figure 6-8 depicts the relationship between the number of scenarios and the improvement in OCP's long-run operational costs compared to a naive baseline of not installing solar panels or batteries and satisfying OCP's energy needs via purchasing electricity from the grid locally at each site. We conclude that both the improvement in operational cost (left) and the investment policy prescribed by the model (right) vary less than 0.2 and 5%, respectively, as we increase the number of scenarios. Therefore, we conclude that our model is stable as we vary the number of scenarios.



(a) (b)

Figure 6-8: The model is insensitive to the number of scenarios. We report the improvement in operational cost (left) and investment policy (right) as we vary the number of scenarios.

6.4.2 Trading Off Investment Costs Against Operational Costs and CO_2 Emissions

In this section, we investigate the potential benefits of our cross-validated model for OCP in terms of both cost savings and emission reductions. Specifically, we explore the relationship between OCP's investment budget in the cross-validated model, and its long-run costs and carbon emissions.

We solve our cross-validated model with an overall investment budget of B for each $B \in \{2, 4, \dots 20\}$ billion MAD. For each budget B, we compute OCP's anticipated operational costs (i.e., the cost of procuring energy from ONEE/NAREVA, plus cost of renting lines, minus profit from energy sold) and expected CO_2 emissions reduction over the 20-year time horizon with 10 reduced scenarios. Combining insights from our cross-validation procedure and sensitivity analysis of Section 6.7 with discussions with the OCP team, we set the RO and DRO hyperparameters to $(\gamma_{\max}, \gamma_C, \gamma_{\text{CLT}}, \delta) = (0.5, 0.5, 0.5, 0.01)$. We compare our cross-validated model against the no-investment baseline which satisfies OCP's energy needs via purchasing electricity from the grid locally at each site, in terms of both improvement in operational costs and emissions reduction, in Figure 6-9. We remark that the baseline's operational costs on the test set is about 35 billion MAD.

Impact of Investment on Operational Costs. Observe that as OCP's investment budget increases, its total savings over the entire 20-year planning horizon increase, but so does the amount of time required for OCP to recover its costs. Therefore, there exists a trade-off between how much OCP invests and how much it can expect to profit from this initiative: investing more capital to a budget of 10 billion MAD, if not more, increases its improvement in operational costs over the entire 20-year horizon by nearly 80%, which translates to approximately 28 billion MAD. However, it also increases the amount of time required for OCP to break even. Therefore, OCP ultimately needs to choose an investment level which balances its attitude towards risk against both its interest in lowering global CO_2 emissions and the profit that it could make from this initiative.

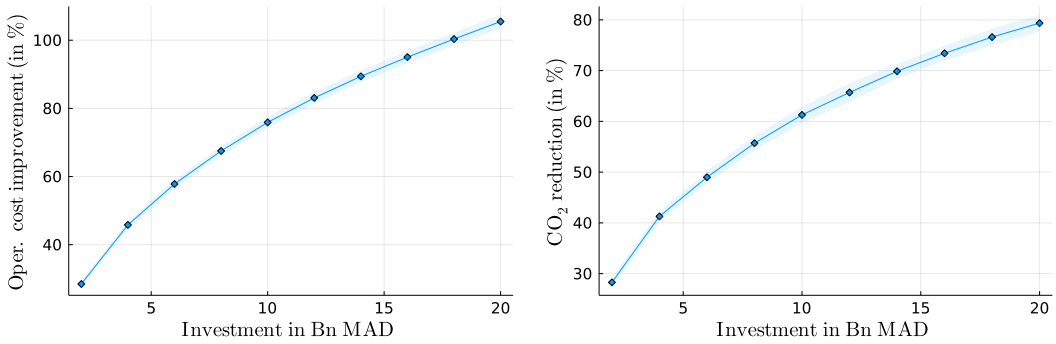


Figure 6-9: Trade-off between overall investment cost and operational costs (left), CO_2 emissions (right). Increasing investment decreases OCP's cost of operations (left) and OCP's carbon emissions (right).

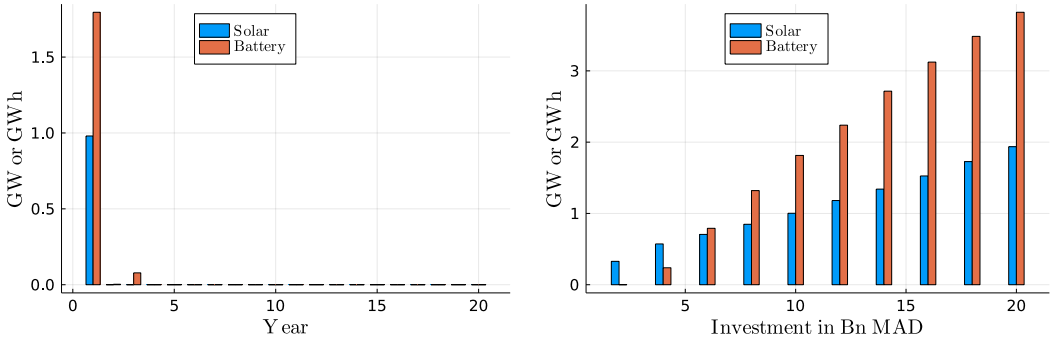
Impact of Investment on CO_2 Emissions. Figure 6-9 (right) investigates this relationship, and demonstrates the value of OCP's investment from an environmental perspective. Namely, with an investment of 10 billion MAD, OCP reduces its emissions by around 60%, while with a more substantial investment of 20 billion MAD, OCP reduces its emissions by a more substantial 80%.

6.4.3 The Model in Action

In this section, we investigate the policy prescribed by our robustified and cross-validated model with a 20-year planning horizon, 10 reduced scenarios, and a 10 billion MAD budget, in order to obtain managerial insights into the structure of optimal solar capacity expansion strategies.

Strategic decisions. Figure 6-10 (left) presents the total prescribed investment in solar generation and batteries by year of the planning horizon. We observe that the model installs the vast majority of all batteries and solar panels in the first year of the planning horizon, and subsequently installs some additional batteries in year three. The significant proportion of solar panels and batteries installed in year one occurs because an investment in renewable energy capacity takes time to recoup, and therefore when planning over a finite time horizon earlier investments in renewable

energy are, generally speaking, more profitable. The subsequent investment in year three reflects two ideas. First, the batteries degrade over time, and therefore installing more as existing ones degrade can be profitable. Second, some existing sources of cogeneration in OCP's network are expected to be phased out over the next five years, and waiting to install their replacement batteries until they are needed allows them to have more of an impact for the rest of the time horizon. Figure 6-10 (right) depicts the prescribed investment in solar generation and batteries aggregated over the entire planning horizon for varying investment budget and reveals that the model prioritizes solar with smaller budget, and includes more batteries as the investment size increases.

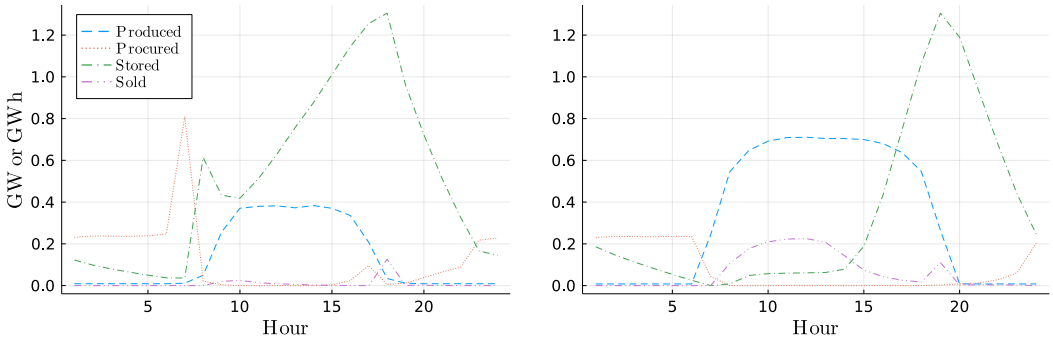


(a) (b)

Figure 6-10: Investment policy prescribed by the model over the entire horizon (aggregated across sites). We report the investment by year (left) and overall investment for increasing budget (right).

Operational Decisions. Figure 6-11 depicts the energy flow in OCP's network in two common reduced scenarios which respectively represent cloudy, low-generation days (Figure 6-11 (left)) and sunny, high-generation days (Figure 6-11 (right)), aggregated across all sites and years in the planning horizon. The installed solar panels produce a large amount of energy during both days–notably more energy during summer–which is used meet demand, refill OCP's batteries, and even occasionally sold to the grid. Moreover, the battery storage levels (reported in green) are kept sufficiently high by the solar energy generated during the day that OCP never purchases energy during the peak or shoulder periods of the day where the wholesale suppliers charge a higher unit price. Interestingly, in both scenarios, OCP purchases some electricity from the grid

in the early morning (when the price of electricity is low), and sells some electricity to the grid several hours later, when the sun is shining and the price is higher. This suggests that the model both anticipates solar generation before it occurs and captures notions of load shifting, or arbitraging prices across time.



(a) (b)

Figure 6-11: Operational policy prescribed by the model for the two most common month-reduced scenario pairs (aggregated across sites and years). We report a low generation scenario (left) and a high generation scenario (right).

6.4.4 Summary of Main Findings

We now summarize the main managerial insights from our numerical results:

- Figure 6-5 in Section 6.7 demonstrates that the use of robust and distributionally robust optimization can reduce the cost of decarbonization by 7% or more, which is significant in the context of a billion dollar investment in a renewable initiative by OCP. In light of recent very significant investments in renewable energy to combat climate change, e.g., $330 billion in new spending by the United States Congress to combat climate change via the Inflation Reduction Act [208], this suggests that robust optimization and other techniques from the Operations Management and Analytics literatures may have a significant role to play in future efforts to decarbonize.

- Figure 6-6 in Section 6.7 demonstrates that increasing the amount of conservatism (by increasing the RO uncertainty set/DRO ambiguity set size) while holding

the investment budget constant increases the net investment in solar panels and decreases the net investment in batteries.

- Figure 6-9 in Section 6.4.2 demonstrates that a 10 billion MAD investment in solar panels and batteries reduces OCP's carbon emissions by approximately 60%, and is profitable over a 20 year planning horizon. Moreover, a 20 billion MAD investment in solar panels and batteries is reduces OCP's carbon emissions by approximately 80% and is also profitable. This suggests that, in addition to a global carbon arbitrage (where decreasing global CO_2 emissions increases global economic output — see [5]), there is—at least sometimes—a local carbon arbitrage, where decarbonizing is profitable for an energy-dependent manufacturer.

- Figure 6-10 in Section 6.4.3 demonstrates that when operating over a finite time horizon, investing in renewable energy generation earlier in the time horizon results in more cost savings for OCP. In a more general setting, these findings suggest that (a) other large manufacturers could consider decarbonization as a viable and potentially cheaper alternative to relying on increasingly scarce carbon-based energy sources, and (b) if they decarbonize, then investing in renewable energy sooner may be more profitable. This is particularly true given the recent surge in global energy prices since the Russian invasion of Ukraine (c.f. [155]).

6.5 Conclusions

We propose a comprehensive methodology developed in collaboration with OCP, one of the world's largest producers of phosphate and phosphate-based products, for partially decarbonizing its production pipeline using solar panels and batteries. Our methodology prescribes both the long-term strategic decisions and the daily tactical decisions that govern OCP's energy planning, and is robust both to the daily uncertainty in solar generation (using RO) and the uncertainty due to climate change

(using DRO). OCP is currently using our approach as the basis of a ten billion MAD (approx. one billion USD) investment, which will simultaneously reduce its carbon emissions by 60% and lower its operational costs over a 20 year planning horizon.

6.6 Technical Proofs

Proof of Lemma 13.

To reformulate constraint (6.7) into its robust counterpart, we first expand the inner minimization problem:

$$\begin{array}{lll}
\min & \displaystyle\sum_{n,h,d,m,y} \bar{z}_n^y v_n^{h,d,m,y} & \\
\text{s.t.} & v_n^{h,d,m,y} = \bar{v}^{h,d} + u_n^{h,d,m,y} \quad \forall h,d,m,y,n & [\boldsymbol{\sigma_1}] \\
& -U_{\max}^{h,d} \le u_n^{h,d,m,y} \le U_{\max}^{h,d} \quad \forall h,d,m,y,n & [\boldsymbol{\phi_1}, \boldsymbol{\chi_1}] \\
& -U_{\mathrm{C}}^{h,d} \le u_n^{h,d,m,y} - u_n^{h-1,d,m,y} \le U_{\mathrm{C}}^{h,d} \quad \forall h,d,m,y,n & [\boldsymbol{\psi_1}, \boldsymbol{\omega_1}] \\
& \displaystyle\sum_{n,h,m,y} u_n^{h,d,m,y} \le U_{\mathrm{CLT}}^{d} \quad \forall d & [\boldsymbol{\tau_1}] \\
& v_n^{h,d,m,y} \ge 0 \quad \forall h,d,m,y,n &
\end{array} \tag{6.14}$$

where, for simplicity, we take $u^{0,d} = u^{24,d}, \forall d$, and denote the dual variable associated with each constraint in square brackets. Problem (6.14)'s dual is:

$$\begin{array}{ll}
\max & \displaystyle\sum_d \left(\sum_{n,h,m,y} \bar{v}^{h,d} \sigma_{1,n}^{h,d,m,y} + U_{\max}^{h,d} \phi_{1,n}^{h,d,m,y} + U_{\max}^{h,d} \chi_{1,n}^{h,d,m,y} + U_{\mathrm{C}}^{h,d} \psi_{1,n}^{h,d,m,y} + U_{\mathrm{C}}^{h,d} \omega_{1,n}^{h,d,m,y} \right) + U_{\mathrm{CLT}}^{d} \tau_1^d \\
\text{s.t.} & \sigma_{1,n}^{h,d,m,y} \le \bar{z}_n^y \quad \forall h,d,m,y,n \\
& -\sigma_{1,n}^{h,d,m,y} - \phi_{1,n}^{h,d,m,y} + \chi_{1,n}^{h,d,m,y} - \psi_{1,n}^{h,d,m,y} + \omega_{1,n}^{h,d,m,y} + \psi_{1,n}^{h+1,d,m,y} - \omega_{1,n}^{h+1,d,m,y} + \tau_1^d = 0 \quad \forall h,d,m,y,n \\
& \phi_{1,n}^{h,d,m,y} \ge 0,\ \chi_{1,n}^{h,d,m,y} \ge 0,\ \psi_{1,n}^{h,d,m,y} \ge 0,\ \omega_{1,n}^{h,d,m,y} \ge 0,\ \tau_1^d \ge 0 \quad \forall h,d,m,y,n
\end{array} \tag{6.15}$$

where we take $\psi_{1,n}^{25,d,m,y} = \psi_{1,n}^{1,d,m,y},\ \omega_{1,n}^{25,d,m,y} = \omega_{1,n}^{1,d,m,y}$. Linear optimization duality then guarantees that we can replace the minimization problem (Problem (6.14)) in constraint (6.7) with its dual (Problem (6.15)), and this yields the claimed result. □

6.7 Extended Experiments

In Table 6.2, we present the percentage improvement in operational costs on the validation set using different robust hyperparameter values. We independently split the data five times, and report the mean and standard deviation of the results.

Table 6.2: Mean and standard deviation of % improvement in operational costs using validation data.

$(\gamma_{\max}, \gamma_C, \gamma_{CLT})$	δ =0.0	δ =0.001	δ =0.01	δ =0.1	δ =1.0
(0.0, 0.0, 0.0)	0.0 (0.0)	6.82 (0.57)	6.82 (0.56)	6.63 (0.51)	5.1 (0.55)
(0.0, 0.0, 0.5)	0.0 (0.0)	6.69 (0.56)	6.82 (0.56)	6.74 (0.51)	5.1 (0.54)
(0.0, 0.0, 1.0)	0.0 (0.0)	6.82 (0.57)	6.82 (0.56)	6.74 (0.51)	5.23 (0.53)
(0.0, 0.5, 0.0)	0.0 (0.0)	6.83 (0.57)	6.82 (0.56)	6.74 (0.51)	5.1 (0.54)
(0.0, 0.5, 0.5)	0.0 (0.0)	6.83 (0.57)	6.82 (0.56)	6.74 (0.51)	5.1 (0.54)
(0.0, 0.5, 1.0)	0.0 (0.0)	6.83 (0.57)	6.82 (0.56)	6.74 (0.51)	5.1 (0.54)
(0.0, 1.0, 0.0)	0.0 (0.0)	6.83 (0.57)	6.82 (0.56)	6.63 (0.51)	5.1 (0.54)
(0.0, 1.0, 0.5)	0.0 (0.0)	6.69 (0.56)	6.82 (0.56)	6.74 (0.51)	5.1 (0.54)
(0.0, 1.0, 1.0)	-0.0 (0.0)	6.82 (0.57)	6.82 (0.56)	6.74 (0.51)	5.23 (0.54)
(0.5, 0.0, 0.0)	0.0 (0.0)	6.83 (0.57)	6.82 (0.56)	6.63 (0.51)	5.1 (0.54)
(0.5, 0.0, 0.5)	0.0 (0.0)	6.69 (0.56)	6.82 (0.56)	6.74 (0.51)	5.1 (0.54)
(0.5, 0.0, 1.0)	0.0 (0.0)	6.82 (0.57)	6.82 (0.56)	6.74 (0.51)	5.23 (0.53)
(0.5, 0.5, 0.0)	0.47 (0.58)	6.71 (0.52)	6.57 (0.47)	6.21 (0.59)	4.15 (0.43)
(0.5, 0.5, 0.5)	0.47 (0.58)	6.55 (0.45)	6.57 (0.47)	5.93 (0.77)	4.16 (0.44)
(0.5, 0.5, 1.0)	0.47 (0.58)	**6.84 (0.34)**	6.57 (0.47)	5.92 (0.77)	4.21 (0.47)
(0.5, 1.0, 0.0)	0.47 (0.58)	6.55 (0.45)	6.57 (0.48)	5.93 (0.78)	4.12 (0.43)
(0.5, 1.0, 0.5)	0.47 (0.58)	6.71 (0.52)	6.57 (0.48)	5.93 (0.78)	4.18 (0.46)
(0.5, 1.0, 1.0)	0.47 (0.58)	6.71 (0.52)	6.55 (0.53)	5.93 (0.78)	4.12 (0.43)
(1.0, 0.0, 0.0)	0.0 (0.0)	6.83 (0.57)	6.82 (0.56)	6.63 (0.51)	5.1 (0.54)
(1.0, 0.0, 0.5)	0.0 (0.0)	6.69 (0.56)	6.82 (0.56)	6.74 (0.51)	5.1 (0.55)
(1.0, 0.0, 1.0)	0.0 (0.0)	6.82 (0.57)	6.82 (0.56)	6.74 (0.51)	5.23 (0.53)
(1.0, 0.5, 0.0)	0.12 (0.19)	6.75 (0.53)	6.62 (0.48)	5.9 (0.94)	0.16 (1.08)
(1.0, 0.5, 0.5)	0.12 (0.19)	6.59 (0.48)	6.62 (0.48)	5.6 (1.05)	0.16 (1.08)
(1.0, 0.5, 1.0)	0.12 (0.19)	6.75 (0.53)	6.62 (0.48)	5.58 (1.07)	0.22 (1.2)
(1.0, 1.0, 0.0)	0.04 (0.13)	6.6 (0.49)	6.64 (0.48)	5.97 (0.88)	-0.58 (1.32)
(1.0, 1.0, 0.5)	0.04 (0.13)	6.6 (0.49)	6.64 (0.48)	5.7 (0.96)	-0.5 (1.47)
(1.0, 1.0, 1.0)	0.04 (0.13)	6.76 (0.54)	6.58 (0.52)	5.7 (0.96)	-0.46 (1.44)

Chapter 7

Conclusions

"Ithaca gave you the marvelous journey.
Without her you wouldn't have set out.
She has nothing left to give you now.
And if you find her poor, Ithaca won't have fooled you.
Wise as you will have become, so full of experience,
you'll have understood by then what these Ithacas mean."

Constantine P. Cavafy

In this thesis, we have focused on the development of analytics, machine learning, and optimization methodologies that contribute towards addressing complexities that are commonly encountered in modern data science problems, including variability, volume, and velocity. Our work presented in Chapters 2 and 3 introduced the framework of slowly varying machine learning, which provides an interpretable tool to deal with the variability characteristic. In Chapter 4, we proposed the backbone method to address the volume characteristic, specifically for supervised machine learning models with a massive number of features. In Chapter 5, we developed a mixed-integer optimization- and machine learning-based approach for the problem of frequency

estimation in data streams, thereby tackling the velocity characteristic. Finally, in Chapter 6, we demonstrated the potential of analytics methodologies to effectively contribute towards combating the climate crisis, through our collaboration with OCP, a leading fertilizer producer, to decarbonize a large part of its production pipeline. Our methodological and applied contributions highlight the capacity of analytics to positively impact critical issues such as climate change and healthcare operations, despite new challenges posed by modern data science problems.

Bibliography

[1] Global covenant of mayors for climate and energy. https://www.globalcovenantofmayors.org/. Accessed February 13, 2023.

[2] Internet live stats. https://www.internetlivestats.com/google-search-statistics. Accessed: 2020-07-01.

[3] Netflow services and applications. Cisco systems white paper (1999) http://www.cisco.com.

[4] Anders Aamand, Piotr Indyk, and Ali Vakilian. Learned frequency estimation algorithms under zipfian distribution. arXiv preprint arXiv:1908.05198, 2019.

[5] Tobias Adrian, Patrick Bolton, and Alissa M Kleinnijenhuis. The great carbon arbitrage. IMF Working Paper, 2022.

[6] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33(01), pages 1418–1426, 2019.

[7] Sina Aghaei, Andres Gomez, and Phebe Vayanos. Learning optimal classification trees: Strong max-flow formulations. arXiv preprint arXiv:2002.09142, 2020.

[8] Amirali Aghazadeh, Ryan Spring, Daniel Lejeune, Gautam Dasarathy, Anshumali Shrivastava, et al. Mission: Ultra large-scale feature selection using count-sketches. In International Conference on Machine Learning, pages 80–88, 2018.

[9] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34(04), pages 3146–3153, 2020.

[10] Shabbir Ahmed, Alan J King, and Gyana Parija. A multi-stage stochastic integer programming approach for capacity expansion under uncertainty. Journal of Global Optimization, 26(1):3–24, 2003.

[11] Shabbir Ahmed and Nikolaos V Sahinidis. An approximation scheme for stochastic integer programs arising in capacity expansion. Operations Research, 51(3):461–471, 2003.

[12] Carlos M Alaíz, Alvaro Barbero, and José R Dorronsoro. Group fused lasso. In International Conference on Artificial Neural Networks, pages 66–73. Springer, 2013.

[13] H. Almuallim and T. Dietterich. Learning boolean concepts in the presence of many irrelevant features. Artificial Intelligence, 69(1-2):279–305, 1994.

[14] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. Journal of Computer and system sciences, 58(1):137–147, 1999.

[15] Tomàs Aluja-Banet and Eduard Nafria. Stability and scalability in decision trees. Computational Statistics, 18(3-4):505–520, 2003.

[16] Edward Anderson and Andy Philpott. Improving sample average approximation using distributional robustness. INFORMS Journal on Optimization, 4(1):90–124, 2022.

[17] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In Proceedings of the forty-seventh annual ACM symposium on Theory of computing, pages 793–801, 2015.

[18] Alper Atamturk and Andres Gomez. Safe screening rules for l0-regression from perspective relaxations. In Hal Daumé III and Aarti Singh, editors, Proceedings of the 37th International Conference on Machine Learning, volume 119 of Proceedings of Machine Learning Research, pages 421–430. PMLR, 13–18 Jul 2020.

[19] Maya Balakrishnan, Kris Ferreira, and Jordan Tong. Improving human-algorithm collaboration: Causes and mitigation of over-and under-adherence. Available at SSRN 4298669, 2022.

[20] Maria-Florina Balcan, Travis Dick, and Tuomas Sandholm. Learning to branch. In International Conference on Machine Learning, 2018.

[21] Chaithanya Bandi and Dimitris Bertsimas. Tractable stochastic analysis in high dimensions via robust optimization. Mathematical Programming, 134(1):23–70, 2012.

[22] Hamsa Bastani, Osbert Bastani, and Wichinpong Park Sinchaisri. Learning best practices: Can machine learning improve human decision-making. In Academy of Management Proceedings, volume 1, page 14006. Academy of Management Briarcliff Manor, NY 10510, 2021.

[23] E. Beale, M. Kendall, and D. Mann. The discarding of variables in multivariate analysis. Biometrika, 54(3-4):357–366, 1967.

[24] Evelyn ML Beale. On minimizing a convex function subject to linear inequalities. Journal of the Royal Statistical Society: Series B (Methodological), 17(2):173–184, 1955.

[25] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM Journal on Imaging Sciences, 2(1):183–202, 2009.

[26] Adrian Becker, Kassem Benabderrazik, Dimitris Bertsimas, Nada Chtinna, Nada El Majdoub, El Miloudi Mahboubi, Driss Lahlou Kitane, Steve Kokkotos, Georgia Mourtzinou, and Ilyas Rakhis. Toward global food security: Transforming OCP through analytics. INFORMS Journal on Applied Analytics, 52(1):90–107, 2022.

[27] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. Robust optimization. Princeton university press, 2009.

[28] Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of uncertain linear programs. Operations Research Letters, 25(1):1–13, 1999.

[29] K. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu. Enlarging the margins in perceptron decision trees. Machine Learning, 41(3):295–313, 2000.

[30] D. Bertsimas and M. Copenhaver. Characterization of the equivalence of robustification and regularization in linear and matrix regression. European Journal of Operational Research, 270(3):931–942, 2018.

[31] D. Bertsimas and J. Dunn. Machine learning under a modern optimization lens. Dynamic Ideas LLC, 2019.

[32] D. Bertsimas, D. Gamarnik, and J. Tsitsiklis. Estimation of time-varying parameters in statistical models: an optimization approach. Machine Learning, 35(3):225–245, 1999.

[33] D. Bertsimas, P. Jaillet, and S. Martin. Online vehicle routing: The edge of optimization in large-scale applications. Operations Research, 67(1):143–162, 2019.

[34] D. Bertsimas, A. King, and R. Mazumder. Best subset selection via a modern optimization lens. The Annals of Statistics, 44(2):813–852, 2016.

[35] D. Bertsimas and B. Van Parys. Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. The Annals of Statistics, 48(1):300–323, 2020.

[36] Dimitris Bertsimas, Ryan Cory-Wright, and Vassilis Digalakis Jr. Decarbonizing OCP. arXiv preprint arXiv:2209.06341, 2022.

[37] Dimitris Bertsimas, Ryan Cory-Wright, and Jean Pauphilet. A unified approach to mixed-integer optimization problems with logical constraints. SIAM Journal on Optimization, 31(3):2340–2367, 2021.

[38] Dimitris Bertsimas, Arthur Delarue, Patrick Jaillet, and Sebastien Martin. The price of interpretability. arXiv preprint arXiv:1907.03419, 2019.

[39] Dimitris Bertsimas and Dick den Hertog. Robust and adaptive optimization. Dynamic Ideas, 2022.

[40] Dimitris Bertsimas and Vassilis Digalakis. Frequency estimation in data streams: Learning the optimal hashing scheme. IEEE Transactions on Knowledge and Data Engineering, 35(2):1541–1553, 2023.

[41] Dimitris Bertsimas and Vassilis Digalakis Jr. The backbone method for ultra-high dimensional sparse machine learning. Machine Learning, 111(6):2161–2212, 2022.

[42] Dimitris Bertsimas and Vassilis Digalakis Jr. Improving stability in decision tree models. arXiv preprint, 2023. To be submitted.

[43] Dimitris Bertsimas, Vassilis Digalakis Jr, and Omar Skali Lami. Machine learning for preventive maintenance at OCP. arXiv preprint, 2023. To be submitted.

[44] Dimitris Bertsimas, Vassilis Digalakis Jr, Michael Linghzi Li, and Omar Skali Lami. Slowly varying regression under sparsity. arXiv preprint arXiv:2102.10773, 2021.

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

[46] Dimitris Bertsimas and Jack Dunn. Machine learning under a modern optimization lens. Dynamic Ideas LLC Charlestown, MA, 2019.

[47] Dimitris Bertsimas, Jack Dunn, and Ivan Paskov. Stable classification. Journal of Machine Learning Research, 23(296):1–53, 2022.

[48] Dimitris Bertsimas, Jack Dunn, Colin Pawlowski, and Ying Daisy Zhuo. Robust classification. INFORMS Journal on Optimization, 1(1):2–34, 2019.

[49] Dimitris Bertsimas and Agni Orfanoudaki. Pricing algorithmic insurance. arXiv preprint arXiv:2106.00839, 2021.

[50] Dimitris Bertsimas, Jean Pauphilet, Bart Van Parys, et al. Sparse regression: Scalable algorithms and empirical performance. Statistical Science, 35(4):555–578, 2020.

[51] Dimitris Bertsimas and Ioana Popescu. Optimal inequalities in probability theory: A convex optimization approach. SIAM Journal on Optimization, 15(3):780–804, 2005.

[52] Dimitris Bertsimas and Melvyn Sim. The price of robustness. Operations Research, 52(1):35–53, 2004.

[53] Dimitris Bertsimas and Melvyn Sim. Tractable approximations to robust conic optimization problems. Mathematical Programming, 107(1):5–36, 2006.

[54] Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. Machine Learning, 110(2):249–277, 2021.

[55] J. Besag, J. York, and A. Mollié. Bayesian image restoration, with two applications in spatial statistics. Annals of the institute of statistical mathematics, 43(1):1–20, 1991.

[56] Lakshminath Bhuvanagiri and Sumit Ganguly. Estimating entropy over data streams. In European Symposium on Algorithms, pages 148–159. Springer, 2006.

[57] John R Birge and Francois Louveaux. Introduction to stochastic programming. Springer Science & Business Media, 2011.

[58] K. Bleakley and J. P. Vert. The group fused lasso for multiple change-point detection. arXiv preprint arXiv:1106.4199, 2011.

[59] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 13(7):422–426, 1970.

[60] Luigi Boffino, Antonio J Conejo, Ramteen Sioshansi, and Giorgia Oggioni. A two-stage stochastic optimization planning framework to decarbonize deeply electric power systems. Energy Economics, 84:104457, 2019.

[61] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine Learning, 3(1):1–122, 2011.

[62] L. Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[63] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and regression trees. Monterey, CA: Wadsworth and Brooks, 1984.

[64] Leo Breiman. Bagging predictors. Machine Learning, 24(2):123–140, 1996.

[65] Leo Breiman. Heuristics of instability and stabilization in model selection. The Annals of Statistics, 24(6):2350–2383, 1996.

[66] Leo Breiman. Statistical modeling: The two cultures. Statistical Science, 16(3):199–231, 2001.

[67] Bénédicte Briand, Gilles R Ducharme, Vanessa Parache, and Catherine Mercat-Rommens. A similarity measure to assess the stability of classification trees. Computational Statistics & Data Analysis, 53(4):1208–1217, 2009.

[68] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. Internet mathematics, 1(4):485–509, 2004.

[69] C. Brunsdon, S. Fotheringham, and M. Charlton. Geographically weighted regression: a method for exploring spatial nonstationarity. Geographical analysis, 28(4):281–298, 1996.

[70] L. Candanedo, V. Feldheim, and D. Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. Energy and buildings, 140:81–97, 2017.

[71] Blase A Carabello. Transcatheter aortic-valve implantation for aortic stenosis in patients who cannot undergo surgery. Current Cardiology Reports, 13(3):173–174, 2011.

[72] Emilio Carrizosa, Cristina Molero-Rio, and Dolores Romero Morales. Mathematical optimization in classification and regression trees. Top, 29(1):5–33, 2021.

[73] E. Casetti. Generating models by the expansion method: applications to geographical research. Geographical analysis, 4(1):81–91, 1972.

[74] Danton S Char, Nigam H Shah, and David Magnus. Implementing machine learning in health care—addressing ethical challenges. The New England Journal of Medicine, 378(11):981, 2018.

[75] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In International Colloquium on Automata, Languages, and Programming, pages 693–703. Springer, 2002.

[76] Fengwei Chen, Arturo Padilla, Peter C. Young, and Hugues Garnier. Data-driven modeling of wireless power transfer systems with slowly time-varying parameters. IEEE Transactions on Power Electronics, 35(11):12442–12456, 2020.

[77] Ningyuan Chen, Ming Hu, and Wenhao Li. Algorithmic decision-making safeguarded by human knowledge. arXiv preprint arXiv:2211.11028, 2022.

[78] P. Chen, C. Tsai, Y. Chen, K. Chou, et al. A linear ensemble of individual and blended models for music rating prediction. In Proceedings of KDD-Cup 2011, pages 21–60, 2012.

[79] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794, 2016.

[80] So-Min Cheong, Kris Sankaran, and Hamsa Bastani. Artificial intelligence for climate change adaptation. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, page e1459, 2022.

[81] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. Journal of the ACM (JACM), 68(1):1–39, 2021.

[82] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. Foundations and Trends in Databases, 4(1–3):1–294, 2012.

[83] Graham Cormode and Marios Hadjieleftheriou. Finding the frequent items in streams of data. Communications of the ACM, 52(10):97–105, 2009.

[84] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms, 55(1):58–75, 2005.

[85] George B Dantzig. Linear programming under uncertainty. Management Science, 1(3-4):197–206, 1955.

[86] Coral Davenport, Justin Gillis, Sewell Chan, and Melissa Eddy. Inside the Paris climate deal. New York Times, 12:15, 2015.

[87] D. De Cock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. Journal of Statistics Education, 19(3), 2011.

[88] Erick Delage and Yinyu Ye. Distributionally robust optimization under moment uncertainty with application to data-driven problems. Operations Research, 58(3):595–612, 2010.

[89] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Overcoming algorithm aversion: People will use imperfect algorithms if they can (even slightly) modify them. Management Science, 64(3):1155–1170, 2018.

[90] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 71–80, 2000.

[91] Royaume du Maroc. Contribution déterminée au niveau national-actualisée. DCN-Maroc, Juin, 2021.

[92] D. Dua and C. Graff. UCI machine learning repository, 2017.

[93] M.A. Duran and I.E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming, 36(3):307–339, 1986.

[94] Kenneth Dwyer and Robert Holte. Decision tree instability and active learning. In Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18, pages 128–139. Springer, 2007.

[95] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. The Annals of Statistics, 32(2):407–499, 2004.

[96] M. Efroymson. Stepwise regression–a backward and forward look. Eastern Regional Meetings of the Institute of Mathematical Statistics, 1966.

[97] Fritz C Eilber, Gerald Rosen, Scott D Nelson, Michael Selch, Frederick Dorey, Jeffery Eckardt, and Frederick R Eilber. High-grade extremity soft tissue sarcomas: factors predictive of local recurrence and its effect on morbidity and mortality. Annals of Surgery, 237(2):218, 2003.

[98] Randall L Eubank. Nonparametric regression and spline smoothing. CRC press, 1999.

[99] J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American Statistical Association, 96(456):1348–1360, 2001.

[100] J. Fan and J. Lv. Sure independence screening for ultrahigh dimensional feature space. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 70(5):849–911, 2008.

[101] J. Fan and J. Lv. Sure independence screening. Wiley StatsRef: Statistics Reference Online, 2018.

[102] J. Fan, R. Samworth, and Y. Wu. Ultrahigh dimensional feature selection: beyond the linear model. Journal of Machine Learning Research, 10:2013–2038, 2009.

[103] J. Fan and R. Song. Sure independence screening in generalized linear models with np-dimensionality. The Annals of Statistics, 38(6):3567–3604, 2010.

[104] Jianqing Fan, Yang Feng, and Rui Song. Nonparametric independence screening in sparse ultra-high-dimensional additive models. Journal of the American Statistical Association, 106(494):544–557, 2011.

[105] Michael Ferris and Andy Philpott. Dynamic risked equilibrium. Operations Research, 70(3):1933–1952, 2022.

[106] Michael C Ferris and Andy Philpott. 100% renewable electricity with storage. Technical report, Electrical Power Optimization Center, University of Auckland, 2019.

[107] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. Mathematical Programming, 66:327–349, 1994.

[108] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1):119–139, 1997.

[109] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software, 33(1):1–22, 2010.

[110] J. Friedman, T. Hastie, and R. Tibshirani. glmnet: Lasso and elastic-net regularized generalized linear models. R package version 4, 2020.

[111] Leonardo Gacitua, Pablo Gallegos, Rodrigo Henriquez-Auba, Álvaro Lorca, Matías Negrete-Pincetic, Daniel Olivares, A Valenzuela, and George Wenzel. A comprehensive review on expansion planning: Models and tools for energy policy analysis. Renewable and Sustainable Energy Reviews, 98:346–360, 2018.

[112] Michele A Gadd, Ephraim S Casper, James M Woodruff, Patricia M McCormack, and Murray F Brennan. Development and treatment of pulmonary metastases in adult patients with extremity soft tissue sarcoma. Annals of Surgery, 218(6):705, 1993.

[113] William B. Gail. Climate's troubling unknown unknowns. The New York Times, April 2019.

[114] D. Gamarnik and I. Zadik. High-dimensional regression with binary coefficients. estimating squared error and a phase transition. arXiv preprint arXiv:1701.04455, 2017.

[115] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Data stream management: processing high-speed data streams. Springer, 2016.

[116] Bernhard Geissler, Ludwig Hermann, Michael C Mew, and Gerald Steiner. Striving toward a circular economy for phosphorus: The role of phosphate rock mining. Minerals, 8(9):395, 2018.

[117] Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. Proceedings of the IEEE, 98(6):937–947, 2010.

[118] Talia Gillis, Bryce McLaughlin, and Jann Spiess. On the fairness of machine-assisted human decisions. arXiv preprint arXiv:2110.15310, 2021.

[119] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning, pages 1093–1103, 2012.

[120] Richard C Grinold. Model building techniques for the correction of end effects in multistage convex programs. Operations Research, 31(3):407–431, 1983.

[121] Allan Grønlund, Kasper Green Larsen, Alexander Mathiasen, Jesper Sindahl Nielsen, Stefan Schneider, and Mingzhou Song. Fast exact k-means, k-medians and bregman divergence clustering in 1d. arXiv preprint arXiv:1701.07204, 2017.

[122] Cheng Guo, Merve Bodur, and Dimitri J Papageorgiou. Generation expansion planning with revenue adequacy constraints. Computers & Operations Research, 142:105736, 2022.

[123] Gurobi Optimization Inc. Gurobi Optimizer Reference Manual, 2016.

[124] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

[125] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. Journal of Machine Learning Research, 3:1157–1182, 2003.

[126] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. Machine learning, 46(1):389–422, 2002.

[127] Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. A comment on "computational complexity of stochastic programming problems". Mathematical Programming, 159(1):557–569, 2016.

[128] T. Hastie, R. Tibshirani, and M. Wainwright. Statistical learning with sparsity: the lasso and generalizations. CRC press, 2015.

[129] Trevor Hastie and Robert Tibshirani. Varying-coefficient models. Journal of the Royal Statistical Society: Series B (Methodological), 55(4):757–779, 1993.

[130] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. The elements of statistical learning: data mining, inference, and prediction, volume 2. Springer, 2009.

[131] H. Hazimeh, R. Mazumder, and A. Saab. Sparse regression at scale: Branch-and-bound rooted in first-order optimization. arXiv preprint arXiv:2004.06152, 2020.

[132] Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. Operations Research, 68(5):1517–1537, 2020.

[133] IBM Watson Health. The 5 vs of big data. `https://web.archive.org/web/20210118085939/https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/`. Accessed February 13, 2023.

[134] H. Henderson and S. Searle. On deriving the inverse of a sum of matrices. Siam Review, 23(1):53–60, 1981.

[135] T.K. Ho. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):832–844, 1998.

[136] R. Hocking and R. Leslie. Selection of the best subset in regression analysis. Technometrics, 9(4):531–540, 1967.

[137] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In International Conference on Learning Representations, 2019.

[138] Rouba Ibrahim, Song-Hee Kim, and Jordan Tong. Eliciting human judgment for prediction algorithms. Management Science, 67(4):2314–2325, 2021.

[139] Interpretable AI. Interpretable AI Documentation, 2020.

[140] Jan O Jansen, Claire Cochran, Dwayne Boyers, Katie Gillies, Robbie Lendrum, Sam Sadek, Fiona Lecky, Graeme MacLennan, and Marion K Campbell. The effectiveness and cost-effectiveness of resuscitative endovascular balloon occlusion of the aorta (reboa) for trauma patients with uncontrolled torso haemorrhage: study protocol for a randomised clinical trial (the uk-reboa trial). Trials, 23(1):1–14, 2022.

[141] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. Science, 349(6245):255–260, 2015.

[142] Nathan Justin, Sina Aghaei, Andres Gomez, and Phebe Vayanos. Optimal robust classification trees. In The AAAI-22 Workshop on Adversarial Machine Learning and Beyond, 2022.

[143] A. Kenney, F. Chiaromonte, and G. Felici. Efficient and effective l_0 feature selection. arXiv preprint arXiv:1808.02526, 2018.

[144] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Advances in Neural Information Processing Systems, pages 6348–6358, 2017.

[145] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[146] Rob Kitchin and Gavin McArdle. What makes big data, big data? exploring the ontological characteristics of 26 datasets. Big Data & Society, 3(1):2053951716631130, 2016.

[147] Burak Kocuk. Conic reformulations for Kullback-Leibler divergence constrained distributionally robust optimization and applications. arXiv preprint arXiv:2007.05966, 2020.

[148] M. Koziarski, B. Krawczyk, and M. Woźniak. The deterministic subspace method for constructing classifier ensembles. Pattern Analysis and Applications, 20(4):981–990, 2017.

[149] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In Proceedings of the 2018 International Conference on Management of Data, pages 489–504, 2018.

[150] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. Learning to optimize join queries with deep reinforcement learning. arXiv preprint arXiv:1808.03196, 2018.

[151] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In Advances in neural information processing systems, pages 1042–1050, 2009.

[152] Mark Last, Oded Maimon, and Einat Minkov. Improving stability of decision trees. International Journal of Pattern Recognition and Artificial Intelligence, 16(02):145–159, 2002.

[153] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet Physics Doklady, volume 10(8), pages 707–710. Soviet Union, 1966.

[154] J. Li, K. Cheng, S. Wang, F. Morstatter, et al. Feature selection: A data perspective. ACM Computing Surveys (CSUR), 50(6):1–45, 2017.

[155] Iana Liadze, Corrado Macchiarelli, Paul Mortimer-Lee, and Patricia Sanchez Juanino. The economic costs of the russia-ukraine conflict. NIESR Policy Paper, 32, 2022.

[156] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1963–1970, 2014.

[157] Guosheng Lin, Chunhua Shen, David Suter, and Anton Van Den Hengel. A general two-step approach to learning-based hashing. In Proceedings of the IEEE international conference on computer vision, pages 2552–2559, 2013.

[158] W. Liu and I. Tsang. Making decision trees feasible in ultrahigh feature and label dimensions. The Journal of Machine Learning Research, 18(81):1–36, 2017.

[159] Olvi L Mangasarian, W Nick Street, and William H Wolberg. Breast cancer diagnosis and prognosis via linear programming. Operations Research, 43(4):570–577, 1995.

[160] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In Proceedings of the ACM Special Interest Group on Data Communication, pages 270–288. 2019.

[161] P Marsh, BA Price, J Holdstock, C Harrison, and MS Whiteley. Deep vein thrombosis (dvt) after venous thermoablation techniques: rates of endovenous heat-induced thrombosis (ehit) and classical dvt after radiofrequency and endovenous laser ablation in a single centre. European Journal of Vascular and Endovascular Surgery, 40(4):521–527, 2010.

[162] Pierre Masse and Robert Gibrat. Application of linear programming to investments in the electric power industry. Management Science, 3(2):149–166, 1957.

[163] F. McSherry and K. Talwar. Mechanism design via differential privacy. In 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07), pages 94–103, 2007.

[164] Bjoern H Menze, B Michael Kelm, Ralf Masuch, Uwe Himmelreich, Peter Bachert, Wolfgang Petrich, and Fred A Hamprecht. A comparison of random forest and its gini importance with standard chemometric methods for the feature selection and classification of spectral data. BMC Bioinformatics, 10(1):1–16, 2009.

[165] Rossella Miglio and Gabriele Soffritti. The comparison between classification trees through proximity measures. Computational Statistics & Data Analysis, 45(3):577–593, 2004.

[166] Zahra Mirzamomen and Mohammad Reza Kangavari. A framework to induce more stable decision trees for pattern classification. Pattern Analysis and Applications, 20:991–1004, 2017.

[167] Jayadev Misra and David Gries. Finding repeated elements. Science of computer programming, 2(2):143–152, 1982.

[168] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In Advances in Neural Information Processing Systems, pages 464–473, 2018.

[169] Michal Moshkovitz, Yao-Yuan Yang, and Kamalika Chaudhuri. Connecting interpretability and robustness in decision trees through separation. In International Conference on Machine Learning, pages 7839–7849. PMLR, 2021.

[170] Ali Mousavi, Ankit B Patel, and Richard G Baraniuk. A deep learning approach to structured signal recovery. In 2015 53rd annual allerton conference on communication, control, and computing (Allerton), pages 1336–1343. IEEE, 2015.

[171] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. Proceedings of the National Academy of Sciences, 116(44):22071–22080, 2019.

[172] Shanmugavelayutham Muthukrishnan. Data streams: Algorithms and applications. Now Publishers Inc, 2005.

[173] B. Natarajan. Sparse approximate solutions to linear systems. SIAM Journal on Computing, 24(2):227–234, 1995.

[174] A. Ng. On feature selection: Learning with exponentially many irrelevant features as training examples. In Proceedings of the Fifteenth International Conference on Machine Learning, pages 404–412. Morgan Kaufmann Publishers Inc., 1998.

[175] Lyu Ni and Fang Fang. Entropy-based model-free feature screening for ultrahigh-dimensional multiclass classification. Journal of Nonparametric Statistics, 28(3):515–530, 2016.

[176] 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.

[177] OCP Group. OCP Sustainability Report 2020. https://ocpsiteprodsa.blob.core.windows.net/media/2021-12/OCP-Sustainability_report_2020-GRI_certified.pdf, 2020. Accessed: 2022-03.

[178] H. Ohlsson, L. Ljung, and S. Boyd. Segmentation of arx-models using sum-of-norms regularization. Automatica, 46(6):1107–1111, 2010.

[179] Asami Okada, Osamu Nakamoto, Maya Komori, Hideki Arimoto, Hiroshi Rinka, and Hiroaki Nakamura. Resuscitative endovascular balloon occlusion of the aorta as an adjunct for hemorrhagic shock due to uterine rupture: a case report. Clinical Case Reports, 5(10):1565, 2017.

[180] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S Sathiya Keerthi. Learning state representations for query optimization with deep reinforcement learning. In Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, pages 1–4, 2018.

[181] T. Palmer and P. Williams. The rumsfeld matrix. In The Climate Demon: Past, Present, and Future of Climate Prediction, pages 193–211. Cambridge University Press, 2021.

[182] Pablo A Parrilo. Semidefinite programming relaxations for semialgebraic problems. Mathematical Programming, 96(2):293–320, 2003.

[183] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[184] Mario VF Pereira and Leontina MVG Pinto. Multi-stage stochastic optimization applied to energy planning. Mathematical Programming, 52(1):359–375, 1991.

[185] Peter CB Phillips. Regression with slowly varying regressors and nonlinear trends. Econometric Theory, pages 557–614, 2007.

[186] Mert Pilanci, Martin J Wainwright, and Laurent El Ghaoui. Sparse learning via boolean relaxations. Mathematical Programming, 151(1):63–87, 2015.

[187] Avner Priel and Boaz Tamir. A vectorial tree distance measure. Scientific Reports, 12(1):5256, 2022.

[188] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In Advances in Neural Information Processing Systems, pages 9661–9670, 2018.

[189] I. Quesada and I. Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. Computers & chemical engineering, 16(10-11):937–947, 1992.

[190] Hamed Rahimian and Sanjay Mehrotra. Distributionally robust optimization: A review. arXiv preprint arXiv:1908.05659, 2019.

[191] M. Redmond and A. Baveja. A data-driven software tool for enabling cooperative information sharing among police departments. European Journal of Operational Research, 141(3):660–678, 2002.

[192] C. Rojas and B. Wahlberg. On change point detection using the fused lasso method. arXiv preprint arXiv:1401.5408, 2014.

[193] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. ACM Computing Surveys (CSUR), 55(2):1–96, 2022.

[194] Ernst Roos and Dick den Hertog. Reducing conservatism in robust optimization. INFORMS Journal on Computing, 32(4):1109–1127, 2020.

[195] J. Schneider, C. Froschhammer, I. Morgenstern, T. Husslein, and J.M. Singer. Searching for backbones—an efficient parallel algorithm for the traveling salesman problem. Computer Physics Communications, 96(2-3):173–188, 1996.

[196] Santiago Akle Serrano. Algorithms for unsymmetric cone optimization and an implementation for problems with the exponential cone. Stanford University, 2015.

[197] William D Shannon and David Banks. Combining classification trees using mle. Statistics in Medicine, 18(6):727–740, 1999.

[198] Alexander Shapiro. On complexity of multistage stochastic programs. Operations Research Letters, 34(1):1–8, 2006.

[199] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. Lectures on stochastic programming: Modeling and theory. SIAM, 3rd edition, 2021.

[200] Vincent G Sigillito, Simon P Wing, Larrie V Hutton, and Kile B Baker. Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest, 10(3):262–266, 1989.

[201] Kavinesh J Singh, Andy B Philpott, and R Kevin Wood. Dantzig-Wolfe decomposition for solving multistage stochastic capacity-planning problems. Operations Research, 57(5):1271–1286, 2009.

[202] Q. Song and F. Liang. A split-and-merge bayesian variable selection approach for ultrahigh dimensional regression. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 77(5):947–972, 2015.

[203] Allen L Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. Operations Research, 21(5):1154–1157, 1973.

[204] WM Stewart, DW Dibb, AE Johnston, and TJ Smyth. The contribution of commercial fertilizer nutrients to food production. Agronomy Journal, 97(1):1–6, 2005.

[205] Steven Stoft. Power system economics: Designing markets for electricity, volume 468. IEEE press Piscataway, 2002.

[206] Mineral Commodity Summaries. US geological survey, 2020. Mineral Commodity Summaries., 2020.

[207] Leandro Utino Taniguchi, Mário Diego Teles Correia, and Fernando Godinho Zampieri. Overwhelming post-splenectomy infection: narrative review of the literature. Surgical Infections, 15(6):686–693, 2014.

[208] Jim Tankersley. Biden signs expansive health, climate and tax law. New York Times, 2022.

[209] Burke T Thompson, Felipe Munera, Stephen M Cohn, Alexandra A MacLean, John Cameron, Luis Rivas, and David Bajayo. Novel computed tomography scan scoring system predicts the need for intervention after splenic injury. Journal of Trauma and Acute Care Surgery, 60(5):1083–1086, 2006.

[210] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1):267–288, 1996.

[211] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(1):91–108, 2005.

[212] R. J. Tibshirani and J. Taylor. The solution path of the generalized lasso. The Annals of Statistics, 39(3):1335–1371, 2011.

[213] Peter Turney. Bias and the quantification of stability. Machine Learning, 20:23–33, 1995.

[214] Bart PG Van Parys, Peyman Mohajerin Esfahani, and Daniel Kuhn. From data to decisions: Distributionally robust optimization is optimal. Management Science, 67(6):3387–3402, 2021.

[215] T. Walsh and J. Slaney. Backbones in optimization and approximation. In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pages 254–259, 2001.

[216] Haizhou Wang and Mingzhou Song. Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming. The R journal, 3(2):29, 2011.

[217] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. IEEE transactions on pattern analysis and machine intelligence, 40(4):769–790, 2017.

[218] Lihong Wang, Qiang Li, Yanwei Yu, and Jinglei Liu. Region compatibility based stability assessment for decision trees. Expert Systems with Applications, 105:112–128, 2018.

[219] X. Wang, D. Dunson, and C. Leng. Decorrelated feature space partitioning for distributed sparse regression. In Advances in Neural Information Processing Systems, pages 802–810, 2016.

[220] Wolfram Wiesemann, Daniel Kuhn, and Melvyn Sim. Distributionally robust convex optimization. Operations Research, 62(6):1358–1376, 2014.

[221] William H Wolberg and Olvi L Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. Proceedings of the national academy of sciences, 87(23):9193–9196, 1990.

[222] William H Wolberg, W Nick Street, and Olvi L Mangasarian. Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. Cancer Letters, 77(2-3):163–171, 1994.

[223] Xiaolin Wu. Optimal quantization by matrix searching. Journal of algorithms, 12(4):663–673, 1991.

[224] M. Wytock. Time-varying linear regression with total variation regularization. 2014.

[225] M. Wytock, S. Sra, and J. Kolter. Fast newton methods for the group fused lasso. In UAI, pages 888–897, 2014.

[226] Weijun Xie and Xinwei Deng. Scalable algorithms for the sparse ridge regression. SIAM Journal on Optimization, 30(4):3359–3386, 2020.

[227] Rui Xin, Chudi Zhong, Zhi Chen, Takuya Takagi, Margo Seltzer, and Cynthia Rudin. Exploring the whole rashomon set of sparse decision trees. arXiv preprint arXiv:2209.08040, 2022.

[228] H. Xu, C. Caramanis, and S. Mannor. Robust regression and lasso. In Advances in Neural Information Processing Systems, pages 1801–1808, 2009.

[229] Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. Journal of Machine Learning Research, 10(7), 2009.

[230] Huan Xu, Constantine Caramanis, and Shie Mannor. Sparse algorithms are not stable: A no-free-lunch theorem. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(1):187–193, 2011.

[231] J. Yang, M. Mahoney, M. Saunders, and Y. Sun. Feature-distributed sparse regression: a screen-and-clean approach. In Advances in Neural Information Processing Systems, pages 2712–2720, 2016.

[232] Tong Yang, Lun Wang, Yulong Shen, Muhammad Shahzad, Qun Huang, Xiaohong Jiang, Kun Tan, and Xiaoming Li. Empowering sketches with machine learning for network measurements. In Proceedings of the 2018 Workshop on Network Meets AI & ML, NetAI'18, page 15–20, New York, NY, USA, 2018. Association for Computing Machinery.

[233] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13), pages 29–42, 2013.

[234] C. Zhang. Nearly unbiased variable selection under minimax concave penalty. The Annals of Statistics, 38(2):894–942, 2010.

[235] Jianping Zhang. Selecting typical instances in instance-based learning. In Machine Learning Proceedings 1992, pages 470–479. Elsevier, 1992.

[236] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal on Computing, 18(6):1245–1262, 1989.

[237] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. Cautionary tales on air-quality improvement in beijing. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 473(2205):20170457, 2017.

[238] Y. Zhou, U. Porwal, C. Zhang, H. Ngo, et al. Parallel feature selection inspired by group testing. In Advances in Neural Information Processing Systems, pages 3554–3562, 2014.

[239] Albrecht Zimmermann. Ensemble-trees: Leveraging ensemble power inside decision trees. In Discovery Science: 11th International Conference, DS 2008, Budapest, Hungary, October 13-16, 2008. Proceedings 11, pages 76–87. Springer, 2008.

[240] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2):301–320, 2005.

[241] Jikai Zou, Shabbir Ahmed, and Xu Andy Sun. Partially adaptive stochastic optimization for electric power generation expansion planning. INFORMS Journal on Computing, 30(2):388–401, 2018.