

ABSTRACT

GAINES, BRIAN RAYMOND. Penalized Estimation in Statistics: Applications & Algorithms. (Under the direction of Eric Chi and Yichao Wu.)

Penalized estimation, also known as regularization, is a modeling framework that has justifiably received a lot of attention in the statistics and machine learning literature over the past twenty years. This modeling approach augments a loss function with a penalty term that can be used to impose structure or prior knowledge on the solution. In this dissertation, we focus on developing methods for two very different statistical tasks that both fit into the penalization estimation framework, which highlights the framework's flexibility.

For the first method, we compare alternative computing strategies for solving the constrained lasso problem. As its name suggests, the constrained lasso extends the widely-used lasso to handle linear constraints, which provide an additional vehicle for incorporating prior information into the model. In addition to quadratic programming, we employ the alternating direction method of multipliers (ADMM) and also derive an efficient solution path algorithm. Through both simulations and real data examples, we compare the different algorithms and provide practical recommendations in terms of efficiency and accuracy for various sizes of data. We also show that, for an arbitrary penalty matrix, the generalized lasso can be transformed to a constrained lasso, while the converse is not true. Thus, our methods can also be used for estimating a generalized lasso, which has wide-ranging applications. Code for implementing the algorithms is freely available in the MATLAB toolbox `SparseReg`.

The other main method focuses on a different area of statistics, clustering. Clustering is a fundamental unsupervised learning technique that aims to discover groups of objects

in a dataset. Biclustering extends clustering to two dimensions where both observations and variables are grouped concurrently, such as simultaneously clustering cancerous tumors and genes or documents and words. Triclustering is then the natural extension of clustering to three dimensions where the data are organized in a three-dimensional array, or tensor. We develop and study a convex formulation of the triclustering problem, which is guaranteed to obtain a unique global minimum. Convex triclustering generates an entire solution path of possible triclusters governed by one tuning parameter, and thus alleviates the need to specify the number of clusters a priori. We extensively study our method in several simulated settings, and also apply it to an online advertising dataset.

© Copyright 2017 by Brian Raymond Gaines

All Rights Reserved

Penalized Estimation in Statistics: Applications & Algorithms

by
Brian Raymond Gaines

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Statistics

Raleigh, North Carolina

2017

APPROVED BY:

Eric Chi
Chair of Advisory Committee

Yichao Wu
Vice-Chair of Advisory Committee

Hua Zhou

Donald Martin

DEDICATION

To my parents.

BIOGRAPHY

The author grew up in the small town of Ossian, Indiana. While Brian was attending nearby Norwell High School (class of 2002), the rural town had its third stop light installed, which obviously made the front page of the local newspaper. After high school Brian first attended college at IPFW, a local satellite campus of Indiana University, for two years. However, Brian long had his sights set on attending the main Indiana University campus since he, like anyone raised well in Indiana, is a huge fan of the Indiana Hoosiers. Brian spent three glorious years in Bloomington before graduating in 2007 with a Bachelor of Arts, double majoring in Economics and Political Science.

After college, Brian spent four amazing years working as a research assistant in the Research Department at the Federal Reserve Bank of Richmond. Brian started working there shortly before the asset-backed commercial paper market began to dry up, but as we all know, correlation is not causation. Working as an RA at the Richmond Fed would have been a great experience even in normal times, but it was especially interesting to be at the Fed during the financial crisis and Great Recession. While at the Richmond Fed, Brian also took several math classes at Virginia Commonwealth University to better prepare himself for the rigors of graduate school. It was during this time that Brian realized his interest in statistics since, in the words of John Tukey, he likes to “play in everyone’s backyard.” Brian then spent the next six years studying statistics at North Carolina State University. Brian was heavily involved in the Department of Statistics, including a stint as the department’s Graduate Student Association president, the organizer of a memorable ski trip, and a member of several sports teams, winning three championships. Shortly after defending his dissertation on 7-27-17, Brian started working at SAS as a Research Statistician Developer, where he joined a small team in Advanced Analytics

R&D that focuses on increasing the point-and-click capabilities of SAS Studio to make SAS easier to use and learn.

ACKNOWLEDGEMENTS

I would like to thank my two main advisors, Eric Chi and Hua Zhou, for all of their guidance, knowledge, and understanding over the last few years. I would also like to thank Yichao Wu and Donald Martin for serving on my committee. Dr. Zhou effectively served as my main advisor for two years and as a co-advisor afterwards, but graduate school policies constrain me from officially recognizing him as such. I am fortunate that I have been able to continue working with him after he moved to UCLA. I very much appreciate his emphasis on software development as well as both reproducibility and professional development. I am also fortunate that Dr. Chi decided to join the NC State faculty and later serve as my advisor. I actually helped recruit him to NCSU, as I felt like he would make a nice addition to my committee, and sure enough, I was right. The fact that he is a relatively young professor is nice because he can relate to what it is like to be the advisee, and has a lot of “lessons learned” to pass along. Working with him has definitely made me a more effective researcher.

There are numerous other current and former NC State staff and faculty members I would like to thank for the knowledge and support they provided me during grad school. I had the pleasure of teaching under Roger Woodard, Reneé Moore, Kevin Gross, and Herle McGowan, all of whom made me a better teacher. I was an instructor for Dr. Woodard nine different times, so I would like to especially thank him for all of his help and advice, as well as the help from the Undergraduate Program Assistant, Dana Derosier. I would also like to acknowledge Dr. Moore, who has continued to be a friend and mentor even after leaving NC State. Her advice and support were critical to my survival of graduate school. I am grateful to Emily Griffith and Duncan X. Lascelles for the opportunity to serve as a statistical consultant for the NC State College of Veterinary Medicine, as well

as the help and advice I received from Justin Post, David Dickey, and Jon Stallings while in that position. Working as the lead statistician on a variety of projects was invaluable for my development as a professional statistician. I would like to thank Pam Arroway, Sujit Ghosh, Howard Bondell, Kim Weems, and Donald Martin for their help and support while serving as co-directors of the graduate program. I also would like to thank Montse Fuentes, Leonard Stefanski, and Dennis Boos for their leadership and contributions to the department. Lastly, I want to thank Alison McCoy for all that she has done for me and others while serving as a second mom to the graduate students.

The one thing that set NC State apart when I was choosing a graduate program was the people, and I was not disappointed. As such, and especially given the size of the program, there are a ton of current and former NCSU graduate students who deserve recognition, including Todd Regh, Logan Lossing, Dana Lossing, Chad Brown, Kathleen Brown, Joe Usset, Danny Modlin, Nick Meyer, Neal Grantham, Josh Day, Susheela Singh, Ali Miller, Marcela Alfaro, So Young Park, Alfredo Farjat, Sarah Hale, Sam Morris, Luke Smith, Brad Turnbull, Andy Beam, Ander Wilson, David Vock, Brian Naughton, Matt Austin, Milo Page, and Andrew Wilcox. Todd Regh was my office mate during the first 2-3 years of the program. His advice, humor, and coffee maker helped me survive the first two years while we logged long hours in the Bureau of Mines. I am also very thankful for my girlfriend and partner in crime, Colleen McKendry, whose support has kept me afloat during these final few years of grad school. She also deserves recognition for keeping me alive when I had a bad case of mononucleosis during my fourth year.

There are several current and former employees of the Richmond Fed that I would like to acknowledge: Alex Wolman, Roy Webb, Kartik Athreya, Jeff Lacker, Ned Prescott, Arantxa Jarque, John Walter, Ray Owens, Bob Hetzel, Tanya Hockaday, Rita Franklin, Lorie Hutchins, Karen Myers, Ailsa Long, Chris Herrington, Kevin Bryan, Sam Henly,

Sarah Watt, Mark House, Jake Blackwood, Devin Reilly, Sabrina Pellerin, Sonya Waddell, Nick Haltom, and Renee Haltom. My arrival at the Richmond Fed could not have been timed any better, not only in terms of being there during the financial crisis but also in terms of the employees I overlapped with, many of whom are still good friends. I would like to especially thank Alex Wolman for taking a chance and giving me the unbelievable opportunity to work at the Fed. I am also especially grateful for Sabrina Pellerin, who is like a long-lost fraternal twin, and among other things was instrumental in helping me realize that NC State was the right place for me. She also helped renew my interest in tennis, a hobby that has been crucial in helping an admitted workaholic take breaks from school and maintain my sanity. It has also been a joy to watch her two awesome kids, Holden and Alyana, grow up.

Last and most importantly, I truly believe that I would not be here without my unbelievably amazing family and friends, whom I am immensely and eternally grateful for. Above all, I would like to thank my parents, Allen and Susan Gaines, for all they have done for me over the years. How they raised me has given me such a huge leg up in life, and for that I will forever be grateful. Over the years I have been fortunate to have been integrated into several families, which has meant the world to me: the Wilsons, DBs, Deckers, Eckerts, and Wyatts. I would like to especially thank Clark and Monica Eckert, Cody Griner, Caleb Decker, Bill Decker, Mark McAfee, Travis White, and Brad Gear. As cliché as it sounds, they are like brothers and sisters to me. I also would like to thank Adam and Kylie McCartney, Keith Koch, Tom Koch, Tracy Koch, David Stead, Josh Gerber, Dustin Weikel, Greg Bunn, Phil McAfee, John Feeney, Connie and Wouter Bolte, Monica and Scott Van Arsdale, Brandie Dafforn, and Jamie Costello. Clark, Monica, Cody, and David were intricate in making IU some of the best years of my life. I am also very honored to be the godfather of Clark and Monica's daughter,

Harper, and it has been really great to watch their kids grow up. More broadly, Monica's entire family deserves special recognition for the support they have provided me during several family vacations and holidays, which was crucial for maintaining sanity during grad school. Terri and George Eckert, as well as Bill and Karen Decker, have been second (and third) parents to me. Caleb, Mark, Travis, and Brad have provided endless good times and laughs over the years. I also would like to thank my Little League baseball coach, Mark De La Garza, for instilling in me a strong work ethic that I have relied on heavily to get to this point. Lastly, I would like to thank Paul Oakenfold, Sound Tribe Sector 9, Michael Franti, Tool, Three 6 Mafia, Lil Wayne, and the Notorious B.I.G. for creating inspirational music.

TABLE OF CONTENTS

LIST OF TABLES	xi
LIST OF FIGURES	xii
Chapter 1 Introduction	1
1.1 Penalized Estimation	1
Chapter 2 Algorithms for Fitting the Constrained Lasso	6
2.1 Introduction	6
2.2 Connection to the Generalized Lasso	10
2.3 Algorithms	15
2.3.1 Quadratic Programming	16
2.3.2 ADMM	17
2.3.3 Path Algorithm	19
2.4 Simulated Examples	25
2.4.1 Sum-to-zero Constraints	26
2.4.2 Non-negativity Constraints	29
2.5 Real Data Applications	31
2.5.1 Global Warming Data	31
2.5.2 Brain Tumor Data	32
2.5.3 Microbiome Data	34
2.6 Conclusion	37
Chapter 3 Generalized Convex Clustering	39
3.1 Introduction	39
3.2 Generalized Convex Clustering	43
3.2.1 Motivation	43
3.2.2 Formulation	44
3.2.3 Results	46
3.3 Discussion	46
Chapter 4 Convex Triclustering	48
4.1 Introduction	48
4.2 Preliminaries: Tensor Background and Notation	50
4.2.1 Tensor Basics	50
4.2.2 Tensor Operations	53
4.2.3 Tensor Decompositions	55
4.3 Literature Review	58
4.3.1 Triclustering	58

4.3.2	Multi-way Clustering	62
4.3.3	Clustering Tensor Objects	63
4.4	A Convex Formulation of Triclustering	66
4.4.1	Formulation	66
4.4.2	Properties	69
4.4.3	Estimation	70
4.5	Practical Considerations	78
4.5.1	Specifying the Weights	78
4.5.2	Choosing ρ	80
4.6	Simulation Studies	82
4.6.1	Cubical Tensors, Checkbox Pattern	85
4.6.2	Rectangular Tensors	93
4.6.3	CANDECOMP/PARAFAC Model	99
4.6.4	Importance of Good Weights	101
4.7	Real Data Application	103
4.8	Discussion	108
	References	110
	Appendices	131
	Appendix A Additional Constrained Lasso Derivations	132
	A.1 Constrained Lasso via Generalized Lasso	132
	A.1.1 Reparameterization	132
	A.1.2 Null-Space Method	134
	A.2 Subgradient Violations	136
	Appendix B Additional Convex Triclustering Simulation Results	140
	B.1 Checkbox Pattern: Balanced Sizes and Homoskedasticity	141
	B.2 Checkbox Pattern: Imbalanced Sizes and Homoskedasticity	147
	B.3 Checkbox Pattern: Balanced Sizes and Heteroskedasticity	151
	B.4 Different Clustering Structures	155
	B.5 Rectangular Tensors	157
	B.6 CP Model, Adjusted Rand Index	163
	B.7 CP Model, Variation of Information	164

LIST OF TABLES

Table 2.1	Solution Path Events	22
Table 4.1	Advertising Data Clustering Results	106

LIST OF FIGURES

Figure 2.1	Global Warming Data. Annual temperature anomalies relative to the 1961-1990 average.	8
Figure 2.2	Simulation 1 Results: Algorithm Runtime. Average algorithm runtime (seconds) plus/minus one standard error for a constrained lasso with sum-to-zero constraints on the coefficients. The solution path’s runtime is averaged across the number of kinks in the path to make the runtime more comparable to the other algorithms estimated at one value of the tuning parameter, $\rho = \rho_{\text{scale}} \cdot \rho_{\text{max}}$	27
Figure 2.3	Simulation 1 Results: Algorithm Accuracy. Objective value error (percent) relative to quadratic programming (QP) for the solution path and ADMM at different values of $\rho_{\text{scale}} = \rho/\rho_{\text{max}}$ for $(n, p) = (500, 1000)$. The results are qualitatively the same for the other combinations of (n, p) considered.	28
Figure 2.4	Simulation 2 Results: Algorithm Runtime. Average algorithm runtime (seconds) plus/minus one standard error for a constrained lasso with non-negativity constraints on the parameters. The solution path’s runtime is averaged across the number of kinks in the path to make the runtime more comparable to the other algorithms which are estimated at one value of the tuning parameter, $\rho = \rho_{\text{scale}} \cdot \rho_{\text{max}}$	30
Figure 2.5	Global Warming Data. Annual temperature anomalies relative to the 1961-1990 average, with trend estimates using isotonic regression and the constrained lasso.	32
Figure 2.6	Brain Tumor Data. Sparse fused lasso estimates on the brain tumor data using both the generalized lasso and the constrained lasso.	35
Figure 2.7	Microbiome Data Solution Paths. Comparison of solution path coefficient estimates on the microbiome dataset using both (a) zero-sum regression and (b) the constrained lasso.	37
Figure 4.1	Fibers of a Third-order Tensor. Source: Kolda & Bader (2009).	52
Figure 4.2	Slices of a Third-order Tensor. Source: Kolda & Bader (2009).	52
Figure 4.3	Rank-one Third-order Tensor. $\mathbf{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$, where the (i, j, k) element is $x_{ijk} = a_i b_j c_k$. Source: Kolda & Bader (2009).	55
Figure 4.4	CP Decomposition. The CP decomposition (4.7) of a third-order tensor. Source: Kolda & Bader (2009).	57
Figure 4.5	Tucker Decomposition. The Tucker decomposition (4.9) of a third-order tensor. Source: Kolda & Bader (2009).	58
Figure 4.6	Tensor with Checkbox Structure. Each mode has two clusters for a total of eight triclusters.	59

Figure 4.7	Example Weights-Induced Edge Graph. A graph with positive weights for $\omega_{m,12}$, $\omega_{m,15}$, and $\omega_{m,34}$ and zero weights between all other nodes, corresponding to the mode- m slices. Source: Chi & Lange (2015).	67
Figure 4.8	Checkerbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error.	87
Figure 4.9	Checkerbox Simulation Results: Impact of Cluster Size Imbalance. Two imbalanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different degrees of cluster size imbalance. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$). Size ratio = 0.5 corresponds to balanced clusters.	89
Figure 4.10	Checkerbox Simulation Results: Impact of Heteroskedasticity. Two balanced clusters per mode with either low or high heteroskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different levels of heteroskedasticity. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$). Noise ratio = 1 corresponds to homoskedastic noise.	91
Figure 4.11	Checkerbox Simulation Results: Impact of Clustering Structure. Different number of balanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different clustering structures, corresponding to either three clusters per mode or two, three, and four clusters along modes one, two, and three. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$).	92
Figure 4.12	Checkerbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with two short modes and one longer mode. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.	95
Figure 4.13	Checkerbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with one short modes and two longer modes. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.	96
Figure 4.14	Checkerbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with short, medium, and long mode lengths. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.	98
Figure 4.15	Factor Matrices for the CP Models.	100

Figure 4.16	CP Model Simulation Results. Two balanced clusters per mode with low homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average triclustering performance plus/minus one standard error for two different data generation approaches. “Bullseye” and “Half Moons” refer to the shape embedded in the factor matrices used to generate the true tensor (Figure 4.15).	101
Figure 4.17	Impact of Convex Triclustering Weights. Comparison of different weights for convex triclustering for clustering a cubical tensor with two balanced clusters per mode and homoskedastic noise with $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different noise levels. TD1 refers to using a Tucker decomposition with the rank chosen using the SCORE algorithm (Section 4.5.1. True uses the true mean tensor \mathbf{U} to construct the weights, while Data uses the noisy data tensor \mathbf{X}	102
Figure 4.18	Advertisement and Publisher Click-Through Rate Biclusters for a Random User. Advertisements are on the y -axis and publishers are on the x -axis. Darker blue corresponds to higher click-through rates for a given device.	108
Figure B.1	Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average adjusted rand index plus/minus one standard error.	141
Figure B.2	Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average variation of information plus/minus one standard error.	142
Figure B.3	Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error.	143
Figure B.4	Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error.	144
Figure B.5	Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 80$. Average adjusted rand index plus/minus one standard error.	145
Figure B.6	Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 80$. Average variation of information plus/minus one standard error.	146

Figure B.7	Checkbox Simulation Results: Impact of Cluster Size Imbalance with Low Noise. Two imbalanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.	147
Figure B.8	Checkbox Simulation Results: Impact of Cluster Size Imbalance with Low Noise. Two imbalanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.	148
Figure B.9	Checkbox Simulation Results: Impact of Cluster Size Imbalance with High Noise. Two imbalanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.	149
Figure B.10	Checkbox Simulation Results: Impact of Cluster Size Imbalance with High Noise. Two imbalanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.	150
Figure B.11	Checkbox Simulation Results: Impact of Heteroskedasticity with Low Noise. Two balanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.	151
Figure B.12	Checkbox Simulation Results: Impact of Heteroskedasticity with Low Noise. Two balanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.	152
Figure B.13	Checkbox Simulation Results: Impact of Heteroskedasticity with High Noise. Two balanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.	153
Figure B.14	Checkbox Simulation Results: Impact of Heteroskedasticity with High Noise. Two balanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.	154

Figure B.15	Checkbox Simulation Results: Impact of Clustering Structure. Different number of balanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different clustering structures, corresponding to either three clusters per mode or two, three, and four clusters along modes one, two, and three. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$).	155
Figure B.16	Checkbox Simulation Results: Impact of Clustering Structure. Different number of balanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different clustering structures, corresponding to either three clusters per mode or two, three, and four clusters along modes one, two, and three. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$).	156
Figure B.17	Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with two short modes and one longer mode. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.	157
Figure B.18	Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with two short modes and one longer mode. Average variation of information plus/minus one standard error for different noise levels and mode lengths.	158
Figure B.19	Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with one short mode and two longer modes. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.	159
Figure B.20	Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with one short mode and two longer modes. Average variation of information plus/minus one standard error for different noise levels and mode lengths.	160
Figure B.21	Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with short, medium, and long mode lengths. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.	161

Figure B.22	Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with short, medium, and long mode lengths. Average variation of information plus/minus one standard error for different noise levels and mode lengths.	162
Figure B.23	CP Model Simulation Results. Two balanced clusters per mode with low homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average adjusted rand index plus/minus one standard error for two different data generation approaches. “Bullseye” and “Half Moons” refer to the shape embedded in the factor matrices used to generate the true tensor (Figure 4.15).	163
Figure B.24	CP Model Simulation Results. Two balanced clusters per mode with low homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average variation of information plus/minus one standard error for two different data generation approaches. “Bullseye” and “Half Moons” refer to the shape embedded in the factor matrices used to generate the true tensor (Figure 4.15).	164

Chapter 1

Introduction

Variable selection and prediction are two fundamental goals in statistics and machine learning. Penalization, also referred to as regularization or shrinkage, is a flexible framework that can be used to achieve either task. In this dissertation, we focus on two methods that are used for very different statistical tasks but both fit into the penalized estimation framework. This highlights the framework's flexibility. In this chapter, we first motivate penalization in the context of regression before discussing the general penalized estimation framework. For a more-detailed treatment of the subject, see Hastie et al. (2009).

1.1 Penalized Estimation

Consider the standard linear regression model,

$$y_i = \mathbf{x}_i^T \boldsymbol{\beta} + \varepsilon_i, \tag{1.1}$$

for $i = 1, \dots, n$, where y_i is the response or outcome variable for the i th observation, $\mathbf{x}_i \in \mathbb{R}^p$ is a vector of predictor or feature variables (covariates) for the i th observation whose j th element is x_{ij} , $\boldsymbol{\beta} \in \mathbb{R}^p$ is the vector of unknown regression coefficients to be estimated, and ε_i is an independently and identically distributed error term. Using matrix notation, we can equivalently write the model (1.1) as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad (1.2)$$

where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix} \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix},$$

and $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_p)^T$. The typical approach to estimating the parameter vector $\boldsymbol{\beta}$ is to perform ordinary least squares (OLS) by solving the minimization problem

$$\begin{aligned} \hat{\boldsymbol{\beta}} &= \arg \min_{\boldsymbol{\beta}} \sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij}\beta_j)^2 \\ &= \arg \min_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2. \end{aligned} \quad (1.3)$$

Assuming that the data (design) matrix \mathbf{X} has full column rank, then (1.3) has the well-known solution $\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$. Despite its wide use and elegant theory, the linear regression model has its shortcomings. Its prediction accuracy can often be improved upon, it does not automatically perform variable selection, and its solution is not unique in the high-dimensional setting when $p > n$ (Hastie et al., 2009).

One way to improve upon the linear model is to augment it with a *penalty* term, $\rho J(\boldsymbol{\beta})$, so the objective function becomes

$$\hat{\boldsymbol{\beta}}(\rho) = \arg \min_{\boldsymbol{\beta}} \left\{ \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho J(\boldsymbol{\beta}) \right\}, \quad (1.4)$$

where $\rho \geq 0$ is the penalization parameter, also called the tuning or regularization parameter, and $J(\boldsymbol{\beta})$ is a user-specified penalty function. Although it is common in the statistical literature to denote the penalty parameter by λ , we instead use ρ since it is common in constrained optimization to use λ to represent the Lagrange multipliers (dual variables). The penalization parameter, ρ , controls the bias-variance trade-off by governing how much weight is put on the penalty function when solving (1.4) for the coefficient estimates. When $\rho = 0$, the penalty term drops out and the estimates match the OLS estimates, but as ρ increases, more emphasis is put on the penalty function. To understand the role played by the penalty function $J(\boldsymbol{\beta})$, consider using the ℓ_1 norm for the penalty, so $J(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1 = \sum_{j=1}^p |\beta_j|$. With this penalty function, the optimization problem becomes

$$\hat{\boldsymbol{\beta}}(\rho) = \arg \min_{\boldsymbol{\beta}} \left\{ \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho \|\boldsymbol{\beta}\|_1 \right\}. \quad (1.5)$$

The model (1.5) is known as the lasso, which stands for the Least Absolute Shrinkage and Selection Operator and was proposed by Tibshirani (1996) in a seminal work. The lasso (1.5) can equivalently be formulated as a constrained regression problem,

$$\begin{aligned} & \underset{\boldsymbol{\beta}}{\text{minimize}} && \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 && (1.6) \\ & \text{subject to} && \|\boldsymbol{\beta}\|_1 \leq t, \end{aligned}$$

where there is a one-to-one correspondence between t and ρ from (1.5) (Tibshirani, 1996). The constrained formulation (1.6) makes the role of the penalty function more clear, as we can see that the constraint shrinks the magnitudes of the estimated regression coefficients as t decreases. One benefit of using the ℓ_1 norm for the penalty is that, due to the geometry of the absolute value function (a sharp kink at zero), the ℓ_1 norm shrinks some coefficients to be exactly zero and thus simultaneously performs continuous variable selection and estimation at the same time (Hastie et al., 2009). For this reason, it is typically recommended to standardize the data matrix \mathbf{X} in penalized regression to put the predictors on the same scale so the penalization is done equitably.

More generally, given a data matrix \mathbf{X} , the goal is to estimate a function $f(\mathbf{X})$ to model or predict the response of interest, \mathbf{y} . The general penalization framework for doing so is given by

$$\underset{f \in \mathcal{H}}{\text{minimize}} \left\{ \sum_{i=1}^n L(y_i, f(\mathbf{x}_i)) + \rho J(f) \right\}, \quad (1.7)$$

where $L(y_i, f(\mathbf{x}_i))$ is a loss function, f is a function belonging to some space of functions \mathcal{H} , $\rho \geq 0$ is a penalization parameter, and $J(f)$ is a penalty functional (Hastie et al., 2009). The loss function measures how well the estimated values fit the observed values. The most commonly used loss function is the squared error loss function, $L(y_i, f(\mathbf{x}_i)) = (y_i - f(\mathbf{x}_i))^2$, which is used for OLS regression (1.3). Other loss functions include absolute error, a negative log-likelihood for generalized linear models, and the hinge loss for support vector machines (Hastie et al., 2009). One way to view the penalty function is that it allows the user to incorporate prior knowledge or structure into the estimation. For example, in the case of the lasso (1.5), the prior knowledge is in the form of sparsity, where the belief is that most of the true coefficients are zero

(Hastie et al., 2009). The lasso essentially spawned a new area of research devoted to tweaking the penalization framework (1.7), typically by modifying the penalty term. As such, several popular methods in statistics can fit into this framework, including ridge regression (Hoerl & Kennard, 1970), the lasso (Tibshirani, 1996), adaptive lasso (Zou, 2006), group lasso (Yuan & Lin, 2006), graphical lasso (Yuan & Lin, 2007; Friedman et al., 2008), generalized lasso (Tibshirani & Taylor, 2011), fused lasso (Tibshirani et al., 2005), ℓ_1 trend filtering (Kim et al., 2009), elastic net (Zou & Hastie, 2005), smoothing splines (O’Sullivan, 1986), convex clustering (Lindsten et al., 2011; Hocking et al., 2011), and convex biclustering (Chi et al., 2017).

Chapter 2

Algorithms for Fitting the Constrained Lasso

2.1 Introduction

Our focus is on estimating the constrained lasso problem (James et al., 2013)

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho \|\boldsymbol{\beta}\|_1 && (2.1) \\ & \text{subject to} && \mathbf{A}\boldsymbol{\beta} = \mathbf{b} \text{ and } \mathbf{C}\boldsymbol{\beta} \leq \mathbf{d}, \end{aligned}$$

where $\mathbf{y} \in \mathbb{R}^n$ is the response vector, $\mathbf{X} \in \mathbb{R}^{n \times p}$ is the data matrix of predictors or covariates, $\boldsymbol{\beta} \in \mathbb{R}^p$ is the vector of unknown regression coefficients, and $\rho \geq 0$ is a tuning parameter that controls the amount of penalization. It is assumed that the constraint matrices, \mathbf{A} and \mathbf{C} , both have full row rank. As its name suggests, the constrained lasso augments the standard lasso (1.5) with linear equality and inequality constraints. While the use of the ℓ_1 penalty enables a user to impose prior knowledge on the coefficient

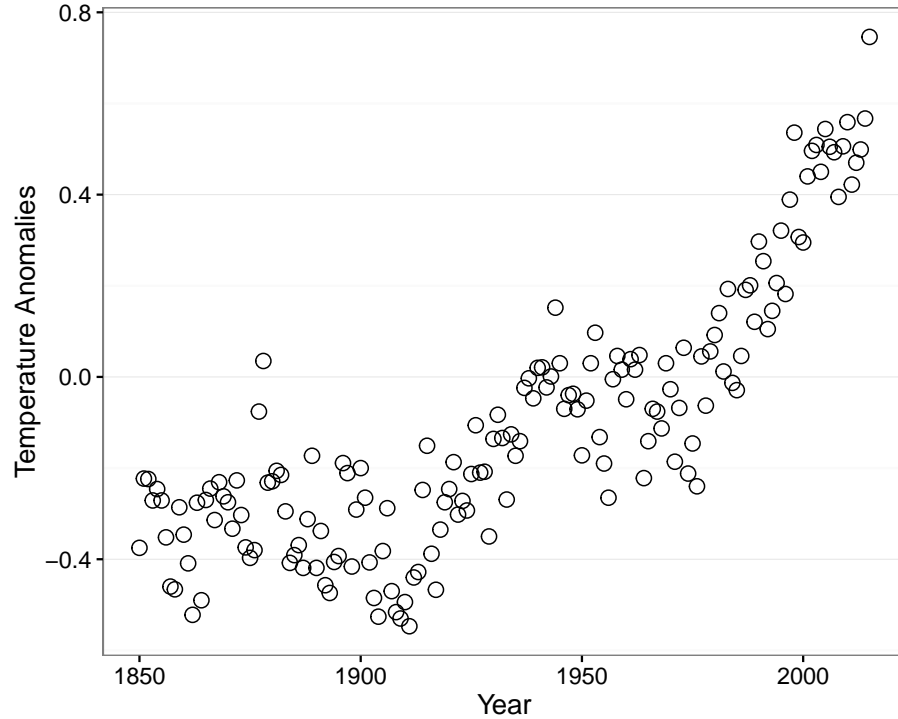


Figure 2.1: Global Warming Data. Annual temperature anomalies relative to the 1961-1990 average.

with the presence of a certain protein in a cell or tissue. The lasso with a sum-to-zero constraint on the coefficients has been used for regression (Shi et al., 2016) and variable selection (Lin et al., 2014) with compositional data as covariates. Compositional data are multivariate data that represent proportions of a whole and thus must sum to one, and are seen in applications such as consumer spending in economics, topic consumption of documents in machine learning, and the human microbiome (Lin et al., 2014). Lastly, simplex constraints were utilized by Huang et al. (2013a) when using the lasso to estimate edge weights in brain networks. Thus, the constrained lasso is a very flexible framework for imposing additional knowledge and structure onto the lasso coefficient estimates.

During the preparation of this chapter, we became aware of unpublished work by

He (2011) that also derived a solution path algorithm for solving the constrained lasso. However, our approach to deriving the path algorithm is completely different and is more in line with the literature on solution path algorithms (Rosset & Zhu, 2007), especially in the presence of constraints (Zhou & Lange, 2013). Additionally, we address how our algorithms can be adapted to work in the high-dimensional setting where $n < p$, which was not done by He (2011). Furthermore, the approach by He (2011) decomposes the parameter vector, $\boldsymbol{\beta}$, into its positive and negative parts, $\boldsymbol{\beta} = \boldsymbol{\beta}^+ - \boldsymbol{\beta}^-$, thus doubling the size of the problem. On the other hand, we work directly with the original coefficient vector at the benefit of computational efficiency and notational simplicity. Another important contribution of our work is the implementation of our algorithms in the `SparseReg` MATLAB toolbox available on Github.

The constrained lasso was also studied by James et al. (2013) in an earlier version of their manuscript on penalized and constrained (PAC) regression. The current PAC regression framework extends (2.1) by using a negative log likelihood for the loss function to also cover generalized linear models (GLMs), and thus is more general than the problem we study. However, the increased generality of their method comes at the cost of computational efficiency. Furthermore, their path algorithm is not a traditional solution path algorithm as it is fit on a pre-specified grid of tuning parameters, which is fundamentally different from our path following strategy. Additionally, we believe the squared error loss function merits additional attention given its widespread use with the ℓ_1 penalty, and also since the constrained lasso is a natural approach to solving constrained least squares problems in the increasingly common high-dimensional setting. Hu et al. (2015a) studied the constrained generalized lasso, which reduces to the constrained lasso when no penalty matrix is included ($\mathbf{D} = \mathbf{I}_p$). However, they do not derive a solution path algorithm but instead develop a coordinate descent algorithm for fixed values of the tuning parameter.

The rest of the chapter is organized as follows. In Section 2.2, we demonstrate a new connection between the constrained lasso and the generalized lasso, which shows that the latter can always be transformed and solved as a constrained lasso, even when the penalty matrix is rank deficient. Given the flexibility of the generalized lasso, this result greatly extends the applicability of our algorithms and results. Various algorithms to solve the constrained lasso, including quadratic programming (QP), the alternating direction method of multipliers (ADMM), and a novel path following algorithm, are derived in Section 2.3. Simulation results that compare the performance of the various algorithms are presented in Section 2.4. The main result from the simulations is that, in terms of runtime the solution path algorithm is more efficient than the other approaches when the coefficient estimates are desired at more than a handful of values of the penalization parameter. Real data examples that highlight the flexibility of the constrained lasso are given in Section 2.5, while Section 2.6 concludes.

2.2 Connection to the Generalized Lasso

Another flexible lasso formulation is the generalized lasso (Tibshirani & Taylor, 2011)

$$\text{minimize} \quad \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho \|\mathbf{D}\boldsymbol{\beta}\|_1, \quad (2.2)$$

where $\mathbf{D} \in \mathbb{R}^{m \times p}$ is a fixed, user-specified regularization matrix. Certain choices of \mathbf{D} correspond to different versions of the lasso, including the original lasso, various forms of the fused lasso, and trend filtering. It has been observed that (2.2) can be transformed to a standard lasso when \mathbf{D} has full row rank (Tibshirani & Taylor, 2011), and it can be transformed to a constrained lasso when \mathbf{D} has full column rank (James et al., 2013). Here we demonstrate that a generalized lasso (2.2) can be transformed to a constrained

lasso (2.1) for an arbitrary penalty matrix \mathbf{D} .

Assume that $\text{rank}(\mathbf{D}) = r$, and consider the singular value decomposition (SVD)

$$\mathbf{D} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{pmatrix} \begin{pmatrix} \mathbf{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{pmatrix} = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T,$$

where $\mathbf{U}_1 \in \mathbb{R}^{m \times r}$, $\mathbf{U}_2 \in \mathbb{R}^{m \times (m-r)}$, $\mathbf{\Sigma}_1 \in \mathbb{R}^{r \times r}$, $\mathbf{V}_1 \in \mathbb{R}^{p \times r}$, and $\mathbf{V}_2 \in \mathbb{R}^{p \times (p-r)}$. We define an augmented matrix

$$\tilde{\mathbf{D}} = \begin{pmatrix} \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T \\ \mathbf{V}_2^T \end{pmatrix} = \begin{pmatrix} \mathbf{U}_1\mathbf{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{pmatrix} = \begin{pmatrix} \mathbf{U}_1\mathbf{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \mathbf{V}^T$$

and use the following change of variables

$$\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\gamma} \end{pmatrix} = \tilde{\mathbf{D}}\boldsymbol{\beta} = \begin{pmatrix} \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T \\ \mathbf{V}_2^T \end{pmatrix} \boldsymbol{\beta}, \quad (2.3)$$

where $\boldsymbol{\alpha} \in \mathbb{R}^m$ and $\boldsymbol{\gamma} \in \mathbb{R}^{p-r}$. Since the matrix \mathbf{V}_2^T forms a basis for the nullspace of \mathbf{D} , $\mathcal{N}(\mathbf{D})$, it has rank $p - r$ and its columns are linearly independent of the columns of \mathbf{D} . Thus, the augmented matrix $\tilde{\mathbf{D}}$ has full column rank, and the new variables $\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\gamma} \end{pmatrix}$

uniquely determine β via

$$\begin{aligned}
\beta &= (\tilde{D}^T \tilde{D})^{-1} \tilde{D}^T \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \left[\mathbf{V} \begin{pmatrix} \Sigma_1 \mathbf{U}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \mathbf{U}_1 \Sigma_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \mathbf{V}^T \right]^{-1} \tilde{D}^T \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \left[\mathbf{V} \begin{pmatrix} \Sigma_1 \mathbf{U}_1^T \mathbf{U}_1 \Sigma_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \mathbf{V}^T \right]^{-1} \tilde{D}^T \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \left[\mathbf{V} \begin{pmatrix} \Sigma_1^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \mathbf{V}^T \right]^{-1} \tilde{D}^T \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \left[(\mathbf{V}^T)^{-1} \begin{pmatrix} \Sigma_1^2 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix}^{-1} \mathbf{V}^{-1} \right] \tilde{D}^T \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \left[\mathbf{V} \begin{pmatrix} \Sigma_1^{-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \mathbf{V}^T \right] \tilde{D}^T \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \mathbf{V} \begin{pmatrix} \Sigma_1^{-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \mathbf{V}^T \mathbf{V} \begin{pmatrix} \Sigma_1 \mathbf{U}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \mathbf{V} \begin{pmatrix} \Sigma_1^{-2} & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \Sigma_1 \mathbf{U}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \mathbf{V} \begin{pmatrix} \Sigma_1^{-2} \Sigma_1 \mathbf{U}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{V}_1 & \mathbf{V}_2 \end{pmatrix} \begin{pmatrix} \Sigma_1^{-1} \mathbf{U}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix} \begin{pmatrix} \alpha \\ \gamma \end{pmatrix} \\
&= \mathbf{V}_1 \Sigma_1^{-1} \mathbf{U}_1^T \alpha + \mathbf{V}_2 \gamma \\
&= \mathbf{D}^+ \alpha + \mathbf{V}_2 \gamma,
\end{aligned}$$

where \mathbf{D}^+ denotes the Moore-Penrose inverse of the matrix \mathbf{D} . However, since the original change of variables is $\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\gamma} \end{pmatrix} = \tilde{\mathbf{D}}\boldsymbol{\beta}$, $\boldsymbol{\beta}$ is uniquely determined if and only if

$$\begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\gamma} \end{pmatrix} \in \mathcal{C}(\tilde{\mathbf{D}}) = \mathcal{C}\left(\begin{pmatrix} \mathbf{U}_1\boldsymbol{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{p-r} \end{pmatrix}\right),$$

if and only if

$$\boldsymbol{\alpha} \in \mathcal{C}(\mathbf{U}_1\boldsymbol{\Sigma}_1) = \mathcal{C}(\mathbf{U}_1) = \mathcal{C}(\mathbf{D}),$$

if and only if

$$\mathbf{U}_2^T \boldsymbol{\alpha} = \mathbf{0}_{m-r},$$

where $\mathcal{C}(\mathbf{D})$ is the column space of the matrix \mathbf{D} . Therefore, the generalized lasso problem (2.2) is equivalent to a constrained lasso problem

$$\begin{aligned} & \underset{\boldsymbol{\alpha}, \boldsymbol{\gamma}}{\text{minimize}} && \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{D}^+\boldsymbol{\alpha} - \mathbf{X}\mathbf{V}_2\boldsymbol{\gamma}\|_2^2 + \rho\|\boldsymbol{\alpha}\|_1 && (2.4) \\ & \text{subject to} && \mathbf{U}_2^T \boldsymbol{\alpha} = \mathbf{0}_{m-r}, \end{aligned}$$

where $\boldsymbol{\gamma}$ remains unpenalized. There are three special cases of interest:

1. When \mathbf{D} has full row rank, $r = m$, the matrix \mathbf{U}_2 is null and the constraint $\mathbf{U}_2^T \boldsymbol{\alpha} = \mathbf{0}_{m-r}$ vanishes, reducing to a standard lasso as observed by Tibshirani & Taylor (2011).
2. When \mathbf{D} has full column rank, $r = p$, the matrix \mathbf{V}_2 is null and the term $\mathbf{X}\mathbf{V}_2\boldsymbol{\gamma}$

drops, resulting in a constrained lasso as observed by James et al. (2013).

3. When \mathbf{D} does not have full rank, $r < \min(m, p)$, the above problem (2.4) can be simplified to a constrained lasso problem only in $\boldsymbol{\alpha}$ by noticing that minimizing (2.4) with respect to $\boldsymbol{\gamma}$ yields

$$\mathbf{XV}_2\hat{\boldsymbol{\gamma}} = \mathbf{P}_{\mathbf{XV}_2}(\mathbf{y} - \mathbf{XD}^+\boldsymbol{\alpha})$$

for any $\boldsymbol{\alpha}$, where $\mathbf{P}_{\mathbf{XV}_2}$ is the orthogonal projection onto the column space $\mathcal{C}(\mathbf{XV}_2)$. Thus, for an arbitrary penalty matrix \mathbf{D} , using the change of variables (2.3) we end up with a constrained lasso problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\alpha}\|_2^2 + \rho\|\boldsymbol{\alpha}\|_1 \\ \text{subject to} \quad & \mathbf{U}_2^T\boldsymbol{\alpha} = \mathbf{0}_{m-r}, \end{aligned}$$

where $\tilde{\mathbf{y}} = (\mathbf{I} - \mathbf{P}_{\mathbf{XV}_2})\mathbf{y}$ and $\tilde{\mathbf{X}} = (\mathbf{I} - \mathbf{P}_{\mathbf{XV}_2})\mathbf{XD}^+$. The solution path $\hat{\boldsymbol{\alpha}}(\rho)$ can be translated back to that of the original generalized lasso problem via the affine transform

$$\begin{aligned} \hat{\boldsymbol{\beta}}(\rho) &= \mathbf{V}_1\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^T\hat{\boldsymbol{\alpha}}(\rho) + \mathbf{V}_2(\mathbf{V}_2^T\mathbf{X}^T\mathbf{XV}_2)^{-1}\mathbf{V}_2^T\mathbf{X}^T[\mathbf{y} - \mathbf{XD}^+\hat{\boldsymbol{\alpha}}(\rho)] \\ &= [\mathbf{I} - \mathbf{V}_2(\mathbf{V}_2^T\mathbf{X}^T\mathbf{XV}_2)^{-1}\mathbf{V}_2^T\mathbf{X}^T\mathbf{X}]\mathbf{D}^+\hat{\boldsymbol{\alpha}}(\rho) \\ &\quad + \mathbf{V}_2(\mathbf{V}_2^T\mathbf{X}^T\mathbf{XV}_2)^{-1}\mathbf{V}_2^T\mathbf{X}^T\mathbf{y}, \end{aligned}$$

where \mathbf{X}^- denotes the generalized inverse of a matrix \mathbf{X} .

Thus, any generalized lasso problem can be reformulated as a constrained lasso, so the algorithms and results presented here are applicable to a large class of problems. However,

it is not always possible to transform a constrained lasso into a generalized lasso, as detailed in Appendix A.1.

2.3 Algorithms

In this section, we derive three different algorithms for estimating the constrained lasso (2.1). Throughout this section, we assume that \mathbf{X} has full column rank, which necessitates that $n > p$. For the increasingly prevalent high-dimensional case where $n < p$, we follow the standard approach in the related literature (Tibshirani & Taylor, 2011; Hu et al., 2015a; Arnold & Tibshirani, 2016) and add a small ridge penalty to the original objective function in (2.1). The problem then becomes

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho \|\boldsymbol{\beta}\|_1 + \frac{\varepsilon}{2} \|\boldsymbol{\beta}\|_2^2 && (2.5) \\ & \text{subject to} && \mathbf{A}\boldsymbol{\beta} = \mathbf{b} \text{ and } \mathbf{C}\boldsymbol{\beta} \leq \mathbf{d}, \end{aligned}$$

where ε is some small constant, such as 10^{-4} . Note that the objective (2.5) can be re-arranged into standard constrained lasso form (2.1)

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{y}^* - (\mathbf{X}^*)\boldsymbol{\beta}\|_2^2 + \rho \|\boldsymbol{\beta}\|_1 && (2.6) \\ & \text{subject to} && \mathbf{A}\boldsymbol{\beta} = \mathbf{b} \text{ and } \mathbf{C}\boldsymbol{\beta} \leq \mathbf{d}, \end{aligned}$$

using the augmented data $\mathbf{y}^* = \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}$ and $\mathbf{X}^* = \begin{pmatrix} \mathbf{X} \\ \sqrt{\varepsilon} \mathbf{I}_p \end{pmatrix}$. The augmented data matrix has full column rank, so the following algorithms can then be applied to the augmented form (2.6). As discussed by Tibshirani & Taylor (2011), this approach is attractive for more than just computational reasons, as the inclusion of the ridge penalty may also

improve predictive accuracy.

Before deriving the algorithms, we first define some notation. For a vector \mathbf{v} and index set \mathcal{S} , let $\mathbf{v}_{\mathcal{S}}$ be the sub-vector of size $|\mathcal{S}|$ containing the elements of \mathbf{v} corresponding to the indices in \mathcal{S} , where $|\cdot|$ denotes the cardinality or size of the index set. Similarly, for a matrix \mathbf{M} and another index set \mathcal{T} , the matrix $\mathbf{M}_{\mathcal{S}\mathcal{T}}$ contains the rows from \mathbf{M} corresponding to the indices in \mathcal{S} and the columns of \mathbf{M} from the indices in \mathcal{T} . We use a colon, $:$, when all indices along one of the dimensions are included. That is, $\mathbf{M}_{\mathcal{S}:$ contains the rows from \mathbf{M} corresponding to \mathcal{S} but all of the columns in \mathbf{M} .

2.3.1 Quadratic Programming

Our first approach is to use quadratic programming to solve the constrained lasso problem (2.1). The key is to decompose $\boldsymbol{\beta}$ into its positive and negative parts, $\boldsymbol{\beta} = \boldsymbol{\beta}^+ - \boldsymbol{\beta}^-$, as the relation $|\boldsymbol{\beta}| = \boldsymbol{\beta}^+ + \boldsymbol{\beta}^-$ handles the ℓ_1 penalty term. By plugging these into (2.1) and adding the additional non-negativity constraints on $\boldsymbol{\beta}^+$ and $\boldsymbol{\beta}^-$, the constrained lasso is formulated as a standard quadratic program of $2p$ variables,

$$\begin{aligned}
& \text{minimize} && \frac{1}{2} \begin{pmatrix} \boldsymbol{\beta}^+ \\ \boldsymbol{\beta}^- \end{pmatrix}^T \begin{pmatrix} \mathbf{X}^T \mathbf{X} & -\mathbf{X}^T \mathbf{X} \\ -\mathbf{X}^T \mathbf{X} & \mathbf{X}^T \mathbf{X} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}^+ \\ \boldsymbol{\beta}^- \end{pmatrix} + \left(\rho \mathbf{1}_{2p} - \begin{pmatrix} \mathbf{X}^T \mathbf{y} \\ -\mathbf{X}^T \mathbf{y} \end{pmatrix} \right)^T \begin{pmatrix} \boldsymbol{\beta}^+ \\ \boldsymbol{\beta}^- \end{pmatrix} \\
& \text{subject to} && \begin{pmatrix} \mathbf{A} & -\mathbf{A} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}^+ \\ \boldsymbol{\beta}^- \end{pmatrix} = \mathbf{b} \\
& && \begin{pmatrix} \mathbf{C} & -\mathbf{C} \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}^+ \\ \boldsymbol{\beta}^- \end{pmatrix} \leq \mathbf{d} \\
& && \boldsymbol{\beta}^+ \geq \mathbf{0}_p, \quad \boldsymbol{\beta}^- \geq \mathbf{0}_p.
\end{aligned}$$

MATLAB's `quadprog` function is able to scale up to $p \sim 10^2$ - 10^3 , while the commercial GUROBI OPTIMIZER is able to scale up to $p \sim 10^3$ - 10^4 .

2.3.2 ADMM

The next algorithm we apply to the constrained lasso problem (2.1) is the alternating direction method of multipliers (ADMM). The ADMM algorithm has experienced renewed interest in statistics and machine learning applications in recent years as it can solve a large class of problems, is often easy to implement, and is amenable to distributed computing; see Boyd et al. (2011) for a recent survey. In general ADMM is an algorithm to solve a problem that features a separable objective but coupling constraints,

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}) + g(\mathbf{z}) \\ & \text{subject to} && \mathbf{M}\mathbf{x} + \mathbf{F}\mathbf{z} = \mathbf{c}, \end{aligned}$$

where $f, g : \mathbb{R}^p \mapsto \mathbb{R} \cup \{\infty\}$ are closed proper convex functions. The idea is to employ block coordinate descent to the augmented Lagrangian function followed by an update of the dual variables $\boldsymbol{\nu}$,

$$\begin{aligned} \mathbf{x}^{(t+1)} &\leftarrow \arg \min_{\mathbf{x}} \mathcal{L}_\tau(\mathbf{x}, \mathbf{z}^{(t)}, \boldsymbol{\nu}^{(t)}) \\ \mathbf{z}^{(t+1)} &\leftarrow \arg \min_{\mathbf{z}} \mathcal{L}_\tau(\mathbf{x}^{(t+1)}, \mathbf{z}, \boldsymbol{\nu}^{(t)}) \\ \boldsymbol{\nu}^{(t+1)} &\leftarrow \boldsymbol{\nu}^{(t)} + \tau(\mathbf{M}\mathbf{x}^{(t+1)} + \mathbf{F}\mathbf{z}^{(t+1)} - \mathbf{c}), \end{aligned}$$

where t is the iteration counter and the augmented Lagrangian is

$$\mathcal{L}_\tau(\mathbf{x}, \mathbf{z}, \boldsymbol{\nu}) = f(\mathbf{x}) + g(\mathbf{z}) + \boldsymbol{\nu}^T(\mathbf{M}\mathbf{x} + \mathbf{F}\mathbf{z} - \mathbf{c}) + \frac{\tau}{2}\|\mathbf{M}\mathbf{x} + \mathbf{F}\mathbf{z} - \mathbf{c}\|_2^2. \quad (2.7)$$

Often it is more convenient to work with the equivalent scaled form of ADMM, which scales the dual variable and combines the linear and quadratic terms in the augmented Lagrangian (2.7). The updates become

$$\begin{aligned}\mathbf{x}^{(t+1)} &\leftarrow \arg \min_{\mathbf{x}} f(\mathbf{x}) + \frac{\tau}{2} \|\mathbf{M}\mathbf{x} + \mathbf{F}\mathbf{z}^{(t)} - \mathbf{c} + \mathbf{u}^{(t)}\|_2^2 \\ \mathbf{z}^{(t+1)} &\leftarrow \arg \min_{\mathbf{z}} g(\mathbf{z}) + \frac{\tau}{2} \|\mathbf{M}\mathbf{x}^{(t+1)} + \mathbf{F}\mathbf{z} - \mathbf{c} + \mathbf{u}^{(t)}\|_2^2 \\ \mathbf{u}^{(t+1)} &\leftarrow \mathbf{u}^{(t)} + \mathbf{M}\mathbf{x}^{(t+1)} + \mathbf{F}\mathbf{z}^{(t+1)} - \mathbf{c},\end{aligned}$$

where $\mathbf{u} = \boldsymbol{\nu}/\tau$ is the scaled dual variable. The scaled form is especially useful in the case where $\mathbf{M} = \mathbf{F} = \mathbf{I}$, as the updates can be rewritten as

$$\begin{aligned}\mathbf{x}^{(t+1)} &\leftarrow \mathbf{prox}_{\tau f}(\mathbf{z}^{(t)} - \mathbf{c} + \mathbf{u}^{(t)}) \\ \mathbf{z}^{(t+1)} &\leftarrow \mathbf{prox}_{\tau g}(\mathbf{x}^{(t+1)} - \mathbf{c} + \mathbf{u}^{(t)}) \\ \mathbf{u}^{(t+1)} &\leftarrow \mathbf{u}^{(t)} + \mathbf{x}^{(t+1)} + \mathbf{z}^{(t+1)} - \mathbf{c},\end{aligned}$$

where $\mathbf{prox}_{\tau f}$ is the proximal mapping of a function f with parameter $\tau > 0$. Recall that the proximal mapping is defined as

$$\mathbf{prox}_{\tau f}(\mathbf{v}) = \arg \min_{\mathbf{x}} \left(f(\mathbf{x}) + \frac{1}{2\tau} \|\mathbf{x} - \mathbf{v}\|_2^2 \right).$$

One benefit of using the scaled form for ADMM is that, in many situations including the constrained lasso, the proximal mappings have simple, closed form solutions, resulting in straightforward ADMM updates. To apply ADMM to the constrained lasso, we identify f as the objective in (2.1) and g as the indicator function of the constraint set $\mathcal{C} = \{\boldsymbol{\beta} \in$

<ol style="list-style-type: none"> 1 Initialize $\boldsymbol{\beta}^{(0)} = \mathbf{z}^{(0)} = \boldsymbol{\beta}^0, \mathbf{u}^{(0)} = \mathbf{0}_p, \tau > 0$; 2 repeat 3 $\boldsymbol{\beta}^{(t+1)} \leftarrow \operatorname{argmin} \frac{1}{2} \ \mathbf{y} - \mathbf{X}\boldsymbol{\beta}\ _2^2 + \frac{1}{2\tau} \ \boldsymbol{\beta} + \mathbf{z}^{(t)} + \mathbf{u}^{(t)}\ _2^2 + \rho \ \boldsymbol{\beta}\ _1$; 4 $\mathbf{z}^{(t+1)} \leftarrow \operatorname{proj}_{\mathcal{C}}(\boldsymbol{\beta}^{(t+1)} + \mathbf{u}^{(t)})$; 5 $\mathbf{u}^{(t+1)} \leftarrow \mathbf{u}^{(t)} + \boldsymbol{\beta}^{(t+1)} + \mathbf{z}^{(t+1)}$; 6 until <i>convergence criterion is met</i>;
--

Algorithm 1: ADMM for solving the constrained lasso (2.1).

$\mathbb{R}^p : \mathbf{A}\boldsymbol{\beta} = \mathbf{b}, \mathbf{C}\boldsymbol{\beta} \leq \mathbf{d}$,

$$g(\boldsymbol{\beta}) = \delta_{\mathcal{C}}(\boldsymbol{\beta}) = \begin{cases} \infty & \boldsymbol{\beta} \notin \mathcal{C} \\ 0 & \boldsymbol{\beta} \in \mathcal{C}. \end{cases}$$

For the updates, $\operatorname{prox}_{\tau f}$ is a regular lasso problem and $\operatorname{prox}_{\tau g}$ is a projection onto the affine space \mathcal{C} (Algorithm 1). The projection onto convex sets is well-studied and, in many applications, can be solved analytically (see Section 15.2 of Lange (2013) for several examples). For situations where an explicit projection operator is not available, the projection can be found by using quadratic programming to solve the dual problem, which has a smaller number of variables.

2.3.3 Path Algorithm

In this section we derive a novel solution path algorithm for the constrained lasso problem (2.1). According to the KKT conditions, the optimal point $\boldsymbol{\beta}(\rho)$ is characterized by the stationarity condition

$$-\mathbf{X}^T[\mathbf{y} - \mathbf{X}\boldsymbol{\beta}(\rho)] + \rho \mathbf{s}(\rho) + \mathbf{A}^T \boldsymbol{\lambda}(\rho) + \mathbf{C}^T \boldsymbol{\mu}(\rho) = \mathbf{0}_p$$

coupled with the linear constraints. Here $\mathbf{s}(\rho)$ is the subgradient $\partial\|\boldsymbol{\beta}\|_1$ with elements

$$s_j(\rho) = \begin{cases} 1 & \beta_j(\rho) > 0 \\ [-1, 1] & \beta_j(\rho) = 0 \\ -1 & \beta_j(\rho) < 0 \end{cases}, \quad (2.8)$$

and $\boldsymbol{\mu}$ satisfies the complementary slackness condition. That is, $\mu_l = 0$ if $\mathbf{c}_l^T \boldsymbol{\beta} < d_l$ and $\mu_l \geq 0$ if $\mathbf{c}_l^T \boldsymbol{\beta} = d_l$.

Along the path we need to keep track of two sets,

$$\mathcal{A} := \{j : \beta_j \neq 0\}, \quad \mathcal{Z}_I := \{l : \mathbf{c}_l^T \boldsymbol{\beta} = d_l\}.$$

The first set indexes the non-zero (active) coefficients and the second keeps track of the set of (binding) inequality constraints on the boundary. Focusing on the active coefficients for the time being, we have the (sub)vector equation

$$\begin{aligned} \mathbf{0}_{|\mathcal{A}|} &= -\mathbf{X}_{:\mathcal{A}}^T(\mathbf{y} - \mathbf{X}_{:\mathcal{A}}\boldsymbol{\beta}_{\mathcal{A}}) + \rho\mathbf{s}_{\mathcal{A}} + \mathbf{A}_{:\mathcal{A}}^T\boldsymbol{\lambda} + \mathbf{C}_{\mathcal{Z}_I\mathcal{A}}^T\boldsymbol{\mu}_{\mathcal{Z}_I} \\ \begin{pmatrix} \mathbf{b} \\ \mathbf{d}_{\mathcal{Z}_I} \end{pmatrix} &= \begin{pmatrix} \mathbf{A}_{:\mathcal{A}} \\ \mathbf{C}_{\mathcal{Z}_I\mathcal{A}} \end{pmatrix} \boldsymbol{\beta}_{\mathcal{A}}, \end{aligned} \quad (2.9)$$

involving dependent unknowns $\boldsymbol{\beta}_{\mathcal{A}}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\mu}_{\mathcal{Z}_I}$, and independent variable ρ . Applying the implicit function theorem to the vector equation (2.9) yields the path following direction

$$\frac{d}{d\rho} \begin{pmatrix} \boldsymbol{\beta}_{\mathcal{A}} \\ \boldsymbol{\lambda} \\ \boldsymbol{\mu}_{\mathcal{Z}_I} \end{pmatrix} = - \begin{pmatrix} \mathbf{X}_{:\mathcal{A}}^T \mathbf{X}_{:\mathcal{A}} & \mathbf{A}_{:\mathcal{A}}^T & \mathbf{C}_{\mathcal{Z}_I\mathcal{A}}^T \\ \mathbf{A}_{:\mathcal{A}} & \mathbf{0} & \mathbf{0} \\ \mathbf{C}_{\mathcal{Z}_I\mathcal{A}} & \mathbf{0} & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{s}_{\mathcal{A}} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \quad (2.10)$$

The right hand side is constant on a path segment as long as the sets \mathcal{A} and \mathcal{Z}_I and the signs of the active coefficients $\mathbf{s}_{\mathcal{A}}$ remain unchanged. This shows that the solution path of the constrained lasso is piecewise linear. The involved matrix is non-singular as long as $\mathbf{X}_{:\mathcal{A}}$ has full column rank and the constraint matrix $\begin{pmatrix} \mathbf{A}_{:\mathcal{A}} \\ \mathbf{C}_{\mathcal{Z}_I\mathcal{A}} \end{pmatrix}$ has linearly independent rows. The stationarity condition restricted to the inactive coefficients

$$-\mathbf{X}_{:\mathcal{A}^c}^T[\mathbf{y} - \mathbf{X}_{:\mathcal{A}}\boldsymbol{\beta}_{\mathcal{A}}(\rho)] + \rho\mathbf{s}_{\mathcal{A}^c}(\rho) + \mathbf{A}_{:\mathcal{A}^c}^T\boldsymbol{\lambda}(\rho) + \mathbf{C}_{\mathcal{Z}_I\mathcal{A}^c}^T\boldsymbol{\mu}_{\mathcal{Z}_I}(\rho) = \mathbf{0}_{|\mathcal{A}^c|}$$

determines

$$\rho\mathbf{s}_{\mathcal{A}^c}(\rho) = \mathbf{X}_{:\mathcal{A}^c}^T[\mathbf{y} - \mathbf{X}_{:\mathcal{A}}\boldsymbol{\beta}_{\mathcal{A}}(\rho)] - \mathbf{A}_{:\mathcal{A}^c}^T\boldsymbol{\lambda}(\rho) - \mathbf{C}_{\mathcal{Z}_I\mathcal{A}^c}^T\boldsymbol{\mu}_{\mathcal{Z}_I}(\rho). \quad (2.11)$$

Thus $\rho\mathbf{s}_{\mathcal{A}^c}$ moves linearly along the path via

$$\frac{d}{d\rho}[\rho\mathbf{s}_{\mathcal{A}^c}] = - \begin{pmatrix} \mathbf{X}_{:\mathcal{A}}^T\mathbf{X}_{:\mathcal{A}^c} \\ \mathbf{A}_{:\mathcal{A}^c} \\ \mathbf{C}_{\mathcal{Z}_I\mathcal{A}^c} \end{pmatrix}^T \frac{d}{d\rho} \begin{pmatrix} \boldsymbol{\beta}_{\mathcal{A}} \\ \boldsymbol{\lambda} \\ \boldsymbol{\mu}_{\mathcal{Z}_I} \end{pmatrix}. \quad (2.12)$$

The inequality residual $\mathbf{r}_{\mathcal{Z}_I^c} := \mathbf{C}_{\mathcal{Z}_I^c\mathcal{A}}\boldsymbol{\beta}_{\mathcal{A}} - \mathbf{d}_{\mathcal{Z}_I^c}$ also moves linearly with gradient

$$\frac{d}{d\rho}\mathbf{r}_{\mathcal{Z}_I^c} = \mathbf{C}_{\mathcal{Z}_I^c\mathcal{A}}\frac{d}{d\rho}\boldsymbol{\beta}_{\mathcal{A}}. \quad (2.13)$$

Together, equations (2.10), (2.12), and (2.13) are used to monitor changes to \mathcal{A} and \mathcal{Z}_I , which can potentially result in kinks in the solution path.

To recap, since the solution path is piecewise linear we only need to monitor the events discussed above that can result in kinks along the path, and then the rest of the path can

be interpolated. A summary of these events is given in the left-hand side of Table 2.1. We perform path following in the decreasing direction from ρ_{\max} towards $\rho = 0$. Let $\beta^{(t)}$ denote the solution at kink t , then the next kink $t + 1$ is identified by the smallest $\Delta\rho$, where $\Delta\rho > 0$ is determined by the conditions listed in the right column of Table 2.1. In addition to monitoring these events along the path, we also need to ensure that the

Table 2.1: Solution Path Events

Event	Monitor
An active coefficient hits 0	$\beta_{\mathcal{A}}^{(t)} - \Delta\rho \frac{d}{d\rho} \beta_{\mathcal{A}}^{(t)} = \mathbf{0}_{ \mathcal{A} }$
An inactive coefficient becomes active	$[\rho^{(t)} \mathbf{s}_{\mathcal{A}^c}^{(t)}] - \Delta\rho \frac{d}{d\rho} [\rho \mathbf{s}_{\mathcal{A}^c}] = \pm(\rho^{(t)} - \Delta\rho) \mathbf{1}_{ \mathcal{A}^c }$
A strict inequality constraint hits the boundary	$\mathbf{r}_{\mathcal{Z}_I^c}^{(t)} - \Delta\rho \frac{d}{d\rho} \mathbf{r}_{\mathcal{Z}_I^c} = \mathbf{0}_{ \mathcal{Z}_I^c }$
An inequality constraint escapes the boundary	$\boldsymbol{\mu}_{\mathcal{Z}_I}^{(t)} - \Delta\rho \frac{d}{d\rho} \boldsymbol{\mu}_{\mathcal{Z}_I} = \mathbf{0}_{ \mathcal{Z}_I }$

subgradient conditions (2.8) remain satisfied. An issue arises when inactive coefficients on the boundary of the subgradient interval are moving too slowly along the path such that their subgradient would escape $[-1, 1]$ at the next kink $t + 1$. To handle this issue, if an inactive coefficient β_j , $j \in \mathcal{A}^c$, with subgradient $s_j = \pm 1$ is moving too slowly, the coefficient is moved to the active set \mathcal{A} and equation (2.10) is recalculated before continuing the path algorithm. See Appendix A.2 for the explicit ranges of $\frac{d}{d\rho} [\rho \mathbf{s}_{\mathcal{A}^c}]$ that need to be monitored and the corresponding derivations.

Initialization

Since we perform path following in the decreasing direction, a starting value for the tuning parameter, ρ_{\max} , is needed. As $\rho \rightarrow \infty$, the solution to the original problem (2.1)

is given by

$$\begin{aligned} & \text{minimize} && \|\boldsymbol{\beta}\|_1 && (2.14) \\ & \text{subject to} && \mathbf{A}\boldsymbol{\beta} = \mathbf{b} \text{ and } \mathbf{C}\boldsymbol{\beta} \leq \mathbf{d}, \end{aligned}$$

which is a standard linear programming problem. We first solve (2.14) to obtain initial coefficient estimates $\boldsymbol{\beta}^0$ and the corresponding sets \mathcal{A} and \mathcal{Z}_I , as well as initial values for the Lagrange multipliers $\boldsymbol{\lambda}^0$ and $\boldsymbol{\mu}^0$. Following Rosset & Zhu (2007), path following begins from

$$\rho_{\max} = \max_j \left| \mathbf{x}_j^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}^0) - \mathbf{A}_{:j}^T \boldsymbol{\lambda}^0 - \mathbf{C}_{\mathcal{Z}_I, j}^T \boldsymbol{\mu}_{\mathcal{Z}_I}^0 \right|, \quad (2.15)$$

and the subgradient is set according to (2.8) and (2.11). As noted by James et al. (2013), this approach can fail when (2.14) does not have a unique solution. For example, consider a constrained lasso with a sum-to-one constraint on the coefficients, $\sum_j \beta_j = 1$. Any elementary vector \mathbf{e}_j , which has a 1 for the j^{th} element and 0 otherwise, satisfies this constraint while also achieving the minimum ℓ_1 norm, resulting in multiple solutions to (2.14). In this case, it is still possible to use (2.14) and (2.15) to identify ρ_{\max} , which is then used in (2.1) to initialize $\boldsymbol{\beta}^0$, \mathcal{A} , \mathcal{Z}_I , $\boldsymbol{\lambda}^0$, and $\boldsymbol{\mu}^0$ via quadratic programming.

Termination

Another practical consideration for implementing the solution path algorithm is a principled approach to terminating the algorithm. To this end, we derive a formula for the degrees of freedom of the constrained lasso. The standard approach in the lasso literature (Efron et al., 2004; Zou et al., 2007; Tibshirani & Taylor, 2011, 2012) is to rely

on the expression for degrees of freedom given by Stein (1981),

$$\text{df}(g) = \text{E} \left[\sum_{i=1}^n \frac{\partial g_i}{\partial y_i} \right], \quad (2.16)$$

where g is a continuous and almost differentiable function, which with $g(y) = \hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ is satisfied in our case (Hu et al., 2015a). In order to apply (2.16), we need to assume that the response is normally distributed,

$$\mathbf{y} \sim N(\boldsymbol{\mu}, \sigma^2 \mathbf{I}_n).$$

As before, we also assume that both constraint matrices, \mathbf{A} and \mathbf{C} , have full row rank, and \mathbf{X} has full column rank. Then, using the results in Hu et al. (2015a) with $\mathbf{D} = \mathbf{I}_p$, for a fixed $\rho \geq 0$ the degrees of freedom are given by

$$\text{df}(\mathbf{X}\hat{\boldsymbol{\beta}}(\rho)) = \text{E} [|\mathcal{A}| - (q + |\mathcal{Z}_I|)], \quad (2.17)$$

where $|\mathcal{A}|$ is the number of active predictors, q is the number of equality constraints, and $|\mathcal{Z}_I|$ is the number of binding inequality constraints. The unbiased estimator for the degrees of freedom is then $|\mathcal{A}| - (q + |\mathcal{Z}_I|)$. This result is intuitive as the degrees of freedom start out as the number of active predictors, and then one degree of freedom is lost for each equality constraint and each binding inequality constraint. Additionally, when there are no constraints, (2.1) becomes a standard lasso problem with degrees of freedom equal to $|\mathcal{A}|$, consistent with the result in Zou et al. (2007). The formula (2.17) is also consistent with results for constrained estimation presented in Zhou & Lange (2013) and Zhou & Wu (2014). We use the degrees of freedom when implementing the

solution path algorithm, as the path terminates once the degrees of freedom equal n . The number of degrees of freedom is also an important measure that is an input for several metrics used for model assessment and selection, such as Mallows' C_p (Mallows, 1973), the Akaike information criterion (AIC) (Akaike, 1974), or the Bayesian information criterion (BIC) (Schwarz et al., 1978). Specifically, these criteria can be plotted along the path as a function of ρ as a technique for selecting the optimal value for the penalization parameter, as alternatives to cross-validation.

2.4 Simulated Examples

To investigate the performance of the various algorithms outlined in Section 2.3 for solving a constrained lasso problem, we consider two simulated examples. For both simulations, we used the three different algorithms discussed in Section 2.3 to solve (2.1). As noted in Section 2.3.2, the ADMM algorithm includes an additional tuning parameter τ , which we fix at $1/n$ based on initial experiments. Additionally, as pointed out in Boyd et al. (2011), the performance of the ADMM method can be greatly impacted by the choice of the algorithm's stopping criteria, which we set to be 10^{-4} for both the absolute and relative error tolerances. We also used a user-defined function handle to solve the subproblem of projecting onto the constraint set for ADMM as this improves efficiency. Two factors of interest in the simulations are the size of the problem, (n, p) , and the value of the penalization tuning parameter, ρ . Four different levels were used for the size factor, (n, p) : (50, 100), (100, 500), (500, 1000), and (1000, 2000). For the latter factor, the values of ρ were calculated as a fraction of the maximum ρ . The fractions, or ρ_{scale} values (i.e., $\rho = \rho_{\text{scale}} \cdot \rho_{\text{max}}$) used in the simulations were 0.2, 0.4, 0.6, and 0.8, to investigate how the degree of penalization impacts algorithm performance. Since the runtimes for quadratic programming and ADMM are for a fixed value of ρ , the total runtime for the

solution path algorithm is averaged across the number of kinks in the path to make the results more comparable. To generate the data for both simulations, the covariates in the data matrix, \mathbf{X} , were generated as independent and identical (iid) standard normal variables, and the response was generated as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon} \sim N(\mathbf{0}_n, \mathbf{I}_n)$. Both simulations used 20 replicates and were conducted in MATLAB using the `SparseReg` toolbox on a computer with an Intel i7-6700 3.4 GHz processor and 32 GB memory. Quadratic programming uses the `GUROBI OPTIMIZER` via the MATLAB interface, while ADMM and the path algorithm are pure MATLAB implementations.

2.4.1 Sum-to-zero Constraints

The first simulation involves a sum-to-zero constraint on the true parameter vector, $\sum_j \beta_j = 0$. Recently, this type of constraint on the lasso has seen increased interest as it has been used in the analysis of compositional data as well as analyses involving any biological measurement analyzed relative to a reference point (Lin et al., 2014; Shi et al., 2016; Altenbuchinger et al., 2016). Written in the constrained lasso formulation (2.1), this corresponds to $\mathbf{A} = \mathbf{1}_p^T$ and $\mathbf{b} = 0$. For this simulation, the true parameter vector, $\boldsymbol{\beta}$, was defined such that the first 25% of the entries are 1, the next 25% of the entries are -1, and the rest of the elements are 0. Thus the true parameter satisfies the sum-to-zero constraint, which we can impose on the estimation using the constraints.

The main results of the simulation are given in Figure 2.2, which plots the average algorithm runtime results across different problem sizes, (n, p) . The results using quadratic programming (QP) and ADMM are each graphed at two values of ρ_{scale} , 0.2 and 0.6. In the graph we can see that the solution path algorithm was faster than the other methods, and its relative performance is even more impressive as the problem size grows. The graph also shows the impact of the tuning parameter, ρ , on both QP and ADMM. QP

performed similarly across both values of ρ_{scale} , but that was not the case for ADMM. At $\rho_{\text{scale}} = 0.6$, ADMM performed very similarly to QP, but ADMM's performance was much worse at smaller values of ρ . Smaller values of ρ correspond to less weight on the ℓ_1 penalty, which results in solutions that are less sparse. The ADMM runtime is also more variable than the other algorithms.

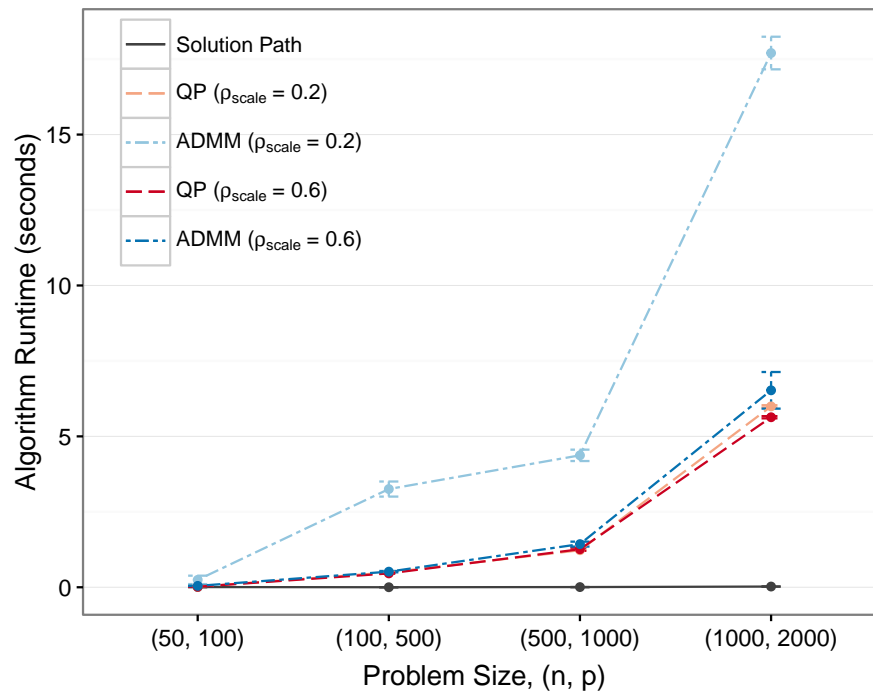


Figure 2.2: Simulation 1 Results: Algorithm Runtime. Average algorithm runtime (seconds) plus/minus one standard error for a constrained lasso with sum-to-zero constraints on the coefficients. The solution path's runtime is averaged across the number of kinks in the path to make the runtime more comparable to the other algorithms estimated at one value of the tuning parameter, $\rho = \rho_{\text{scale}} \cdot \rho_{\text{max}}$.

While algorithm runtime is the metric of primary interest, a fast algorithm is not of much use if it is woefully inaccurate. Figure 2.3 plots the objective value error relative

to QP for the solution path and ADMM for $(n, p) = (500, 1000)$. The results from the other problem sizes are qualitatively similar and are thus omitted. From Figure 2.3, we see that the solution path algorithm is not only efficient but also accurate. On the other hand, the accuracy of ADMM decreases as ρ increases. Part of this is to be expected given that the convergence tolerance used for ADMM is less stringent than the one used for QP. However, the magnitude of these errors is probably not of practical importance.

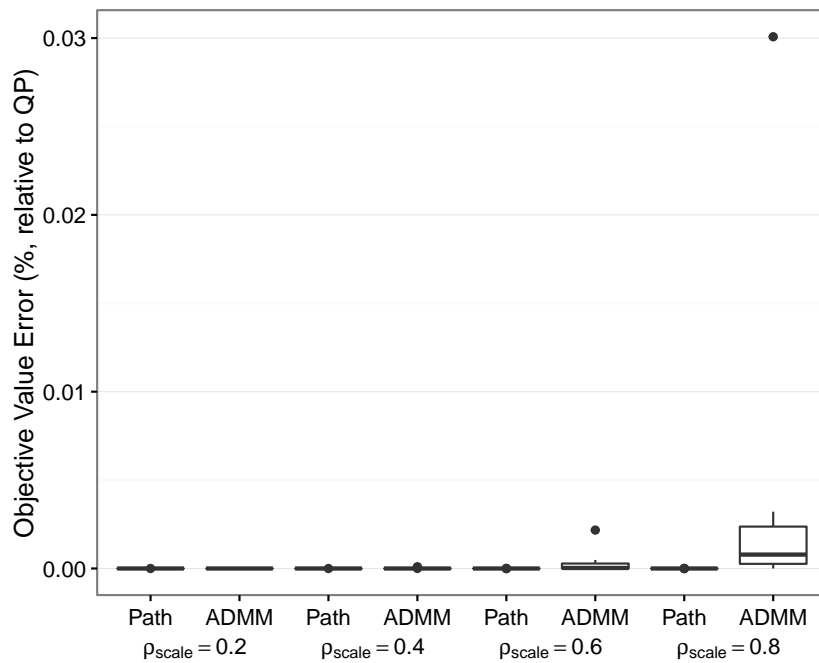


Figure 2.3: Simulation 1 Results: Algorithm Accuracy. Objective value error (percent) relative to quadratic programming (QP) for the solution path and ADMM at different values of $\rho_{\text{scale}} = \rho/\rho_{\text{max}}$ for $(n, p) = (500, 1000)$. The results are qualitatively the same for the other combinations of (n, p) considered.

2.4.2 Non-negativity Constraints

The second simulation involves the positive lasso mentioned in Section 2.1, as to our knowledge it is the most common version of the constrained lasso that has appeared in the literature. Also referred to as the non-negative lasso, as its name suggests it constrains the lasso coefficient estimates to be non-negative. In the constrained lasso formulation (2.1), the positive lasso corresponds to the constraints $\mathbf{C} = -\mathbf{I}_p$ and $\mathbf{d} = \mathbf{0}_p$. For each problem size, the true parameter vector was defined as

$$\beta_j = \begin{cases} j, & j = 1, \dots, 10 \\ 0, & j = 11, \dots, p \end{cases},$$

so the true coefficients obey the constraints and the constrained lasso allows us to incorporate this prior knowledge into the estimation.

Figure 2.4 is a graph of the average runtime for each algorithm for the different problem sizes considered. As with simulation 1, the results for quadratic programming (QP) and ADMM are graphed at two different values of ρ , corresponding to $\rho_{\text{scale}} = \rho/\rho_{\text{max}} \in \{0.2, 0.6\}$ to also demonstrate the impact of ρ on the estimation time. One noteworthy result is that ADMM fared better relative to QP as the problem size grew and was faster than QP for the two larger sizes under investigation, whereas ADMM had runtimes that were comparable or slower than QP in the first simulation. We expected ADMM to outperform QP for larger problems, which happened more quickly in this setting since the inclusion of p inequality constraints notably increased the complexity of the problem. As with the first simulation, another thing that stands out in the results is the strong performance of the solution path algorithm, which generally outperformed the

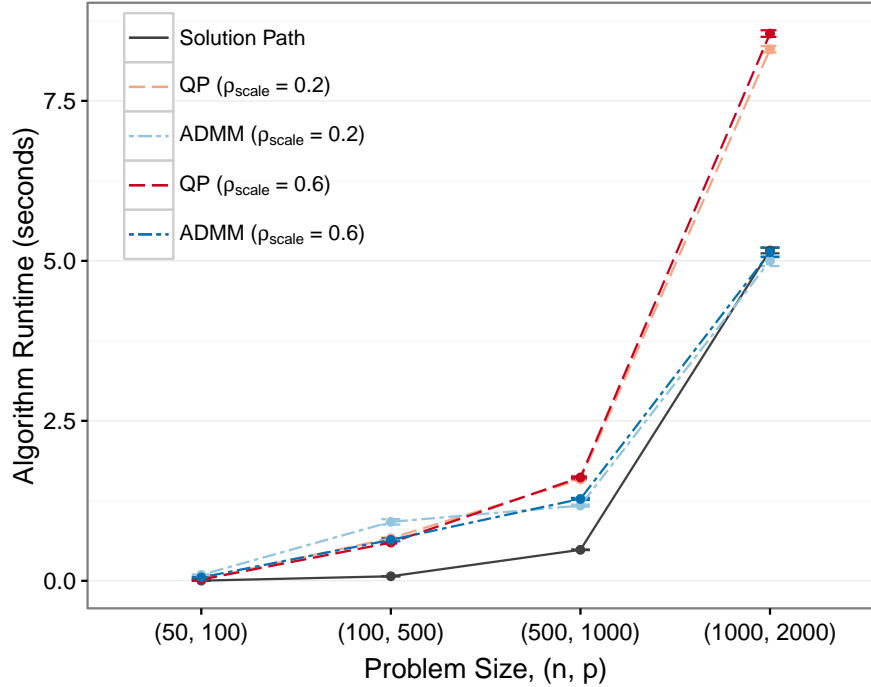


Figure 2.4: Simulation 2 Results: Algorithm Runtime. Average algorithm runtime (seconds) plus/minus one standard error for a constrained lasso with non-negativity constraints on the parameters. The solution path’s runtime is averaged across the number of kinks in the path to make the runtime more comparable to the other algorithms which are estimated at one value of the tuning parameter, $\rho = \rho_{\text{scale}} \cdot \rho_{\text{max}}$.

other two methods. However, for $(n, p) = (1000, 2000)$, ADMM and the solution path performed similarly, partly due to the initialization of the path algorithm hampering its performance as the problem size grows. In terms of accuracy, the objective value errors relative to QP for the solution path and ADMM were negligible and are thus omitted.

2.5 Real Data Applications

2.5.1 Global Warming Data

For our first application of the constrained lasso on a real dataset, we revisit the global temperature data presented in Section 2.1, which was provided by Jones et al. (2016). The dataset consists of annual temperature anomalies from 1850 to 2015, relative to the average for 1961-90. As mentioned, there appears to be a monotone trend to the data over time, so it is natural to want to incorporate this information when estimating the trend. Wu et al. (2001) achieved this on a previous version of the dataset by using isotonic regression, which is given by

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{y} - \boldsymbol{\beta}\|_2^2 && (2.18) \\ & \text{subject to} && \beta_1 \leq \cdots \leq \beta_n, \end{aligned}$$

where $\mathbf{y} \in \mathbb{R}^n$ is the monotonic data series of interest and $\boldsymbol{\beta} \in \mathbb{R}^n$ is a monotonic sequence of coefficients. The lasso analog of isotonic regression, which adds an ℓ_1 penalty term to (2.18), can be estimated by the constrained lasso (2.1) using the constraint matrix

$$\mathbf{C} = \begin{pmatrix} 1 & -1 & & & & \\ & 1 & -1 & & & \\ & & \ddots & \ddots & & \\ & & & & \ddots & \ddots \\ & & & & & 1 & -1 \end{pmatrix}$$

and $\mathbf{d} = \mathbf{0} \in \mathbb{R}^{p-1}$. In this formulation, isotonic regression is a special case of the constrained lasso with $\rho = 0$. This is verified by Figure 2.5, which plots the constrained lasso fit at $\rho = 0$ with the estimates using isotonic regression.

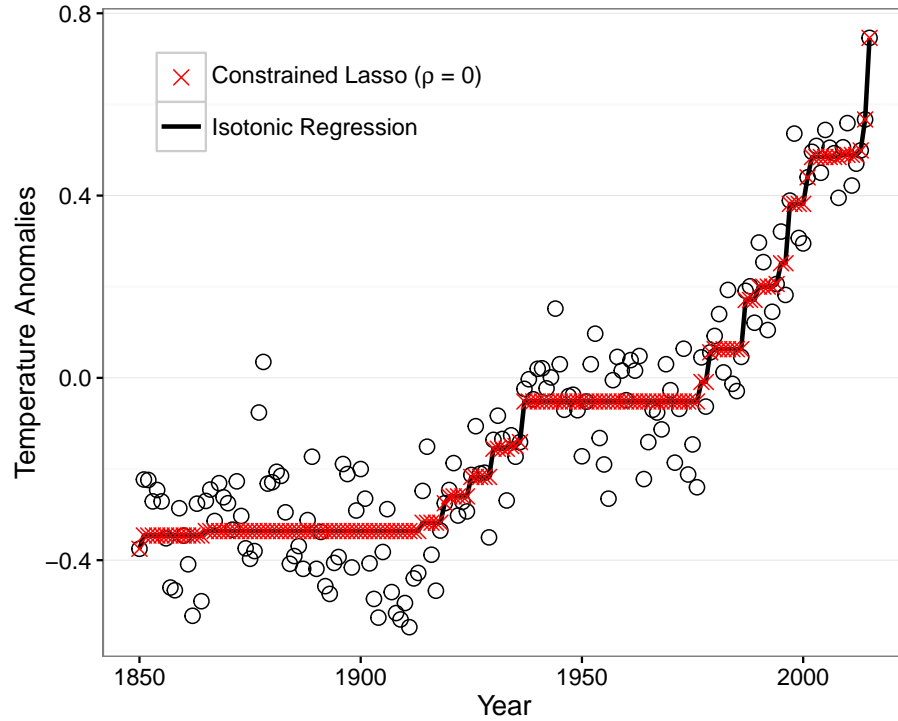


Figure 2.5: Global Warming Data. Annual temperature anomalies relative to the 1961-1990 average, with trend estimates using isotonic regression and the constrained lasso.

2.5.2 Brain Tumor Data

Our second application of the constrained lasso uses a version of the comparative genomic hybridization (CGH) data from Bredel et al. (2005) that was modified and studied by Tibshirani & Wang (2008), which can be seen in Figure 2.6. The dataset contains CGH measurements from 2 glioblastoma multiforme (GBM) brain tumors. CGH

array experiments are used to estimate each gene's DNA copy number by obtaining the \log_2 ratio of the number of DNA copies of the gene in the tumor cells relative to the number of DNA copies in the reference cells. Mutations to cancerous cells result in amplifications or deletions of a gene from the chromosome, so the goal of the analysis is to identify these gains or losses in the DNA copies of that gene (Michels et al., 2007). Tibshirani & Wang (2008) proposed using the sparse fused lasso to approximate the CGH signal by a sparse, piecewise constant function in order to determine the areas with non-zero values, as positive (negative) CGH values correspond to possible gains (losses). The sparse fused lasso (Tibshirani et al., 2005) is given by

$$\text{minimize} \quad \frac{1}{2} \|\mathbf{y} - \boldsymbol{\beta}\|_2^2 + \rho_1 \|\boldsymbol{\beta}\|_1 + \rho_2 \sum_{j=2}^p |\beta_j - \beta_{j-1}|, \quad (2.19)$$

where the additional penalty term encourages the estimates of neighboring coefficients to be similar, resulting in a piecewise constant function. This modification of the lasso was originally termed the fused lasso, but in line with Tibshirani & Taylor (2011) we refer to (2.19) as the sparse fused lasso to distinguish it from the related problem that does not include the sparsity-inducing ℓ_1 norm on the coefficients. Regardless, the sparse fused

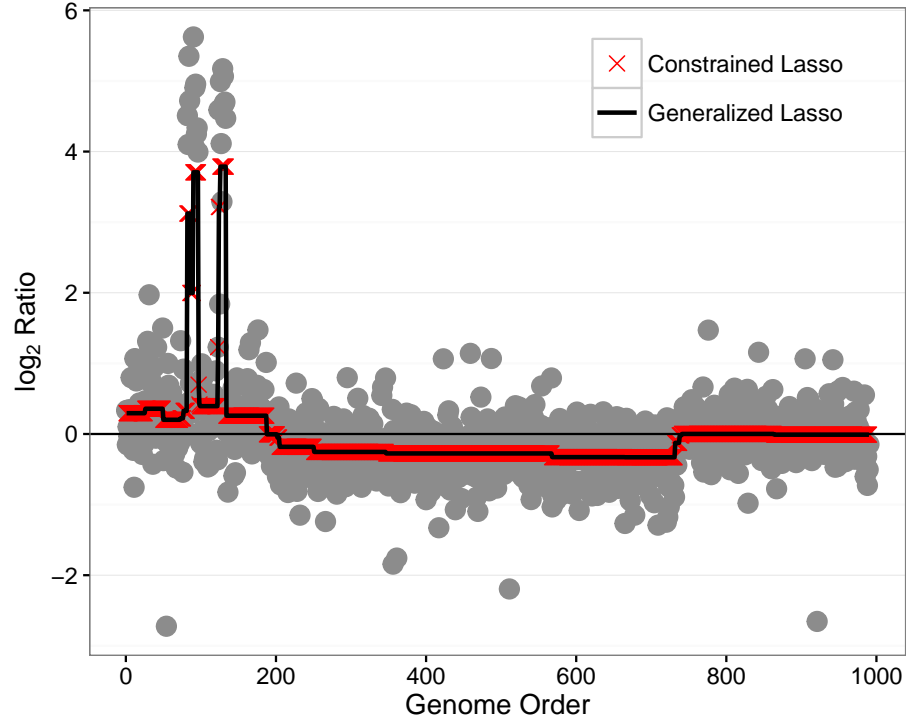


Figure 2.6: Brain Tumor Data. Sparse fused lasso estimates on the brain tumor data using both the generalized lasso and the constrained lasso.

from sample to sample, often the counts are normalized to represent the relative abundance of each bacterium, resulting in compositional data, which are proportions that sum to one. Motivated by this, regression (Shi et al., 2016) and variable selection (Lin et al., 2014) tools for compositional covariates have been developed, which amount to imposing sum-to-zero constraints on the lasso.

Altenbuchinger et al. (2016) built on this work by demonstrating that a sum-to-zero constraint is useful anytime the normalization of data relative to some reference point results in proportional data, as is often the case in biological applications, since the analysis using the constraint is insensitive to the choice of the reference. Altenbuchinger et al. (2016) derived a coordinate descent algorithm for the elastic net with a zero-sum

constraint,

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho \left(\alpha \|\boldsymbol{\beta}\|_1 + \frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2 \right) && (2.20) \\ & \text{subject to} && \sum_j \beta_j = 0, \end{aligned}$$

but the focus of their analysis, which they refer to as *zero-sum regression*, corresponds to $\alpha = 1$, for which (2.20) reduces to the constrained lasso (2.1). Altenbuchinger et al. (2016) applied zero-sum regression to a microbiome dataset from Weber et al. (2015) to demonstrate zero-sum regression’s insensitivity to the reference point, which was not the case for the regular lasso. The data contains the microbiome composition of patients undergoing allogeneic stem cell transplants (ASCT) as well as their urinary levels of 3-indoxyl sulfate (3-IS), a metabolite of the organic compound indole that is produced in the colon and liver. ASCT patients are at high risk for acute graft-versus-host disease and other infectious complications, which have been associated with the composition of the microbiome and the absence of protective microbiota-born metabolites in the gut (Taur et al., 2012; Holler et al., 2014; Murphy & Nguyen, 2011). One such protective substance is indole, which is a byproduct when gut bacteria breaks down the amino acid tryptophan (Weber et al., 2015).

Of interest, then, is to identify a small subset of the microbiome composition associated with 3-IS levels, as the presence of relatively more indole-producing bacteria in the intestines is expected to result in higher levels of 3-IS in urine. ASCT patients receive antibiotics that kill gut bacteria, but with a better understanding of which bacteria produce indole, antibiotics that spare those bacteria could be used instead (Altenbuchinger et al., 2016). The dataset itself contains information on 160 bacteria genera from 37 patients. The bacteria counts were \log_2 -transformed and normalized to have a constant

average across samples. Figure 2.7 plots the coefficient estimate solution paths, $\hat{\beta}(\rho)$, as a function of $\|\hat{\beta}(\rho)\|_1$ using both zero-sum regression and the constrained lasso. As can be seen in the graphs, the coefficient estimates are nearly indistinguishable except for some very minor differences, which are a result of the slightly different formulations of the two problems. Since this is a case where $n < p$, a small ridge penalty is added to the constrained lasso objective function (2.5) as discussed in Section 2.3, but unlike (2.20), the weight on the ℓ_2 penalty does not vary across ρ .

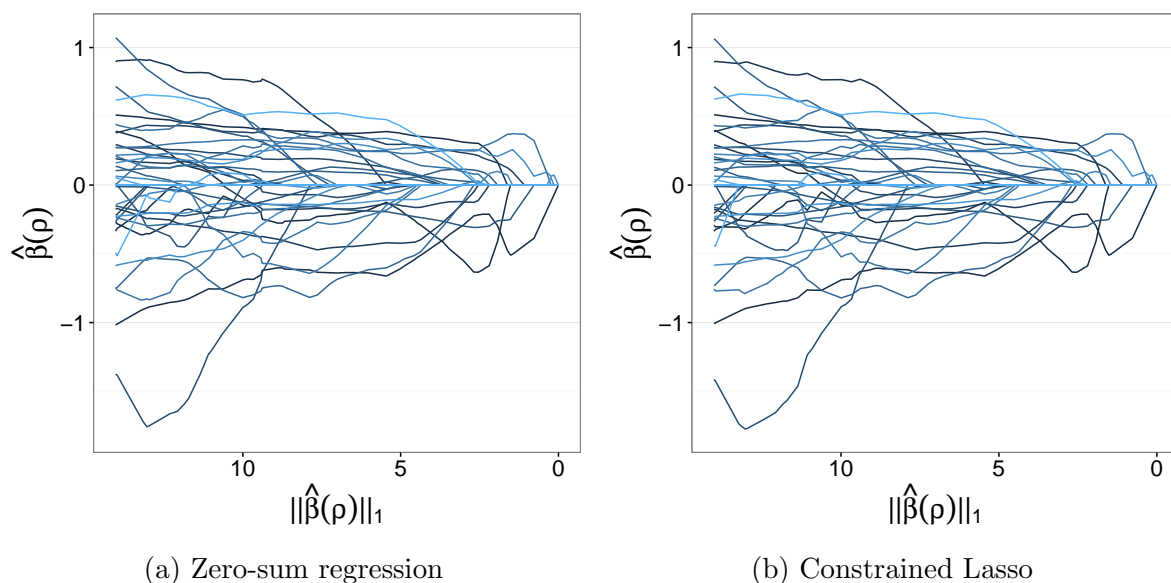


Figure 2.7: Microbiome Data Solution Paths. Comparison of solution path coefficient estimates on the microbiome dataset using both (a) zero-sum regression and (b) the constrained lasso.

2.6 Conclusion

We have studied the constrained lasso problem, in which the original lasso problem is expanded to include linear equality and inequality constraints. As we have discussed

and demonstrated through real data applications, as well as other examples cited from the literature, the constraints allow users to impose prior knowledge on the coefficient estimates. Additionally, we have shown that another flexible lasso variant, the generalized lasso, can always be reformulated and solved as a constrained lasso, which greatly enlarges the trove of problems the constrained lasso can solve.

We derived and compared three different algorithms for computing the constrained lasso solutions as a function of the penalization parameter ρ : quadratic programming (QP), the alternating direction method of multipliers (ADMM), and a novel derivation of a solution path algorithm. Through simulated examples, it was shown that the solution path algorithm outperforms the other methods in terms of estimation time, without sacrificing accuracy. For modest problem sizes, QP remained competitive with ADMM, but ADMM became relatively more efficient as the problem size increased. However, the performance of ADMM is more sensitive to the particular value of ρ . MATLAB code to implement these algorithms is available in the `SparseReg` toolbox on Github.

There are several possible extensions that have been left for future research. The efficiency of the solution path algorithm may be able to be improved by using the sweep operator (Goodnight, 1979) to update (2.10) along the path, as was done in related work by Zhou & Lange (2013). It also may be of interest to extend the algorithms presented here to more general formulations of the constrained lasso. All of the algorithms can be extended to handle general convex loss functions, such as a negative log likelihood function for a generalized linear model extension, which was already studied by James et al. (2013) using a modified coordinate descent algorithm. In this case, an extension of the solution path algorithm could be tracked by solving a system of ordinary differential equations (ODE) as in Zhou & Wu (2014).

Chapter 3

Generalized Convex Clustering

3.1 Introduction

In this chapter, we switch gears and turn our focus for the rest of the dissertation to clustering. Clustering is a widely-used exploratory data analysis technique that seeks to organize a set of objects in a dataset into groups or clusters. The goal is to partition the data so that objects in the same cluster are similar to one another according to some criterion, while objects in different groups are dissimilar. For example, a company may want to cluster its customers as a means to advertise more effectively. It is standard to represent the data as a matrix, $\mathbf{X} \in \mathbb{R}^{n \times p}$, where the rows correspond to the n observations or examples and the columns are the p features or variables. The goal of clustering is then to identify subgroups within the observations or rows of the data matrix \mathbf{X} . In the marketing example, the rows represent different customers while the columns could include things like income, demographic information, and spending habits.

Due to advances in technology and the explosion in data availability over the past two decades, clustering is an especially useful tool as a first step in exploring and analyz-

ing data. As such, clustering has been extensively studied and there are several popular methods, such as k -means, hierarchical clustering, spectral clustering, and Gaussian mixture models (Hastie et al., 2009). Although useful, many of the existing approaches have their flaws. For example, since the majority of clustering methods are cast as non-convex optimization problems, they are solved using greedy algorithms that can become stuck at a local, suboptimal minimum (Hocking et al., 2011; Lindsten et al., 2011; Chi & Lange, 2015; Chi et al., 2017). Recall that a real-valued function $f(\mathbf{x})$ on \mathbb{R}^p is *convex* if

$$f(\alpha\mathbf{x} + (1 - \alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1 - \alpha)f(\mathbf{y}) \quad (3.1)$$

for any $\alpha \in [0, 1]$ and for all \mathbf{x}, \mathbf{y} (Boyd & Vandenberghe, 2004). Additionally, the clustering solution tends to be unstable, both with respect to the algorithm’s initialization and to small changes in the data (Hocking et al., 2011; Lindsten et al., 2011; Chi et al., 2017). Some of the methods also require that the number of clusters, k , be specified beforehand. However, k is typically unknown, especially since clustering is often an exploratory task (Lindsten et al., 2011; Chi & Lange, 2015).

Motivated by these shortcomings of the popular clustering methods, Lindsten et al. (2011) and Hocking et al. (2011) both developed a convex relaxation of k -means and hierarchical clustering. To cluster n data points, $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^p$, corresponding to the rows in the data matrix \mathbf{X} , Lindsten et al. (2011) and Hocking et al. (2011) proposed minimizing the convex objective function

$$\frac{1}{2} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{u}_i\|_2^2 + \rho \sum_{i < j} \omega_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_q, \quad (3.2)$$

where $\mathbf{u}_i \in \mathbb{R}^p$ is the centroid, or cluster center, for the i th data point \mathbf{x}_i , $\rho \geq 0$ is the

penalization parameter, and $\omega_{ij} = \omega_{ji}$ is a nonnegative weight. The first term is the loss function that measures how well the centroid estimates match the data while the second term is a penalty that shrinks together the centroids of the i th and j th data points. Thus we see that convex clustering (3.2) fits into the penalized estimation framework (1.7) discussed in Chapter 1.

The penalization parameter, ρ , controls the extent to which the centroid estimates are fused together via the norm penalty. Lindsten et al. (2011) considered $q \geq 1$ for the norm but primarily used $q = 2$, Hocking et al. (2011) considered $q \in \{1, 2, \infty\}$, and Chi & Lange (2015) worked with an arbitrary norm. In this dissertation, we focus on the ℓ_2 norm for the penalty ($q = 2$). For $\rho = 0$, the penalty term disappears and (3.2) is minimized by $\hat{\mathbf{u}}_i = \mathbf{x}_i$ for $i = 1, \dots, n$, so each data point is in its own cluster. As ρ increases, centroid estimates are fused together as the penalty term carries more weight. This fusion is used to determine the cluster assignments, as two data points \mathbf{x}_i and \mathbf{x}_j are assigned to the same cluster if $\hat{\mathbf{u}}_i = \hat{\mathbf{u}}_j$. Eventually, for a large enough ρ , the points merge into a single cluster. Thus this formulation produces an entire solution path of clustering results, similar in spirit to hierarchical clustering (Hocking et al., 2011; Chen et al., 2015). This is a nice feature of the framework as it alleviates the need to specify the number of clusters a priori as is required by some clustering methods such as k -means. Tan & Witten (2015) proposed using a BIC-type metric for choosing the optimal ρ , which will be discussed more in Section 4.5.2.

The user-specified weights can refine the clustering while also allowing the user to incorporate prior information into the solution (Chen et al., 2015). To increase convex clustering's usability, Chen et al. (2015) extensively investigated the impact of the weights

on the clustering results. They recommended constructing the weights as

$$w_{ij} = \iota_{\{i,j\}}^k \exp(-\phi \|\mathbf{x}_i - \mathbf{x}_j\|_2^2), \quad (3.3)$$

where $\iota_{\{i,j\}}^k$ is an indicator function that equals 1 if the j th data point is among data point i 's k -nearest neighbors (or vice versa) and 0 otherwise, and the second term is the Gaussian kernel that intuitively places a higher weight on data points that are closer together. The weights (3.3) merge together two recommendations originally provided by Lindsten et al. (2011) (k -nearest neighbors) and Hocking et al. (2011) (Gaussian kernel). Inducing sparsity in the weights via k -nearest neighbors not only improves the quality of the clustering results but also improves computational efficiency by reducing the number of terms in the penalty (Chen et al., 2015; Chi & Lange, 2015). The non-negative scale parameter in the Gaussian kernel, ϕ , controls the rate at which the pressure to fuse is applied to the i th and j th data points, while k controls the connectivity of the data points. Since clustering is an exploratory task, Chen et al. (2015) recommended first using different values of k with $\phi = 0$, and then ϕ can be varied to reveal more subtle details once k has been determined.

Since its inception, additional efforts have been made to extend and study convex clustering. Chen et al. (2015) modified convex clustering to be able to handle missing data directly in the algorithm instead of relying on an imputation method. An additional ℓ_1 penalty on the centroid estimates was incorporated into the model by Wang et al. (2017) to extend convex clustering to the high-dimensional setting where the number of features exceeds the number of data points ($p > n$). Chi & Lange (2015) exploited the convexity of the formulation to adopt two algorithms for efficiently computing the convex clustering solution path. Convex clustering has also been extended to two dimensions for performing

biclustering where the rows and columns of the data matrix \mathbf{X} are simultaneously shrunk together (Chi et al., 2017). To achieve this, Chi et al. (2017) introduced an additional penalty term that also fuses together the column centroids.

The statistical properties of convex clustering were studied by Tan & Witten (2015), but their results were not very favorable. However, a shortcoming of both their theoretical and empirical results is that they assume uniform weights in the penalty term, i.e., $\omega_{ij} = 1$ for all i, j . As discussed later (Section 4.6.4) and noted in the literature (Chen et al., 2015; Chi & Lange, 2015; Chi et al., 2017), well-constructed weights, such as (3.3), play an important role in the quality of the clustering results. Currently, there are no theoretical results for convex clustering that incorporate the use of sparse weights, either the oracle weights or those given by (3.3). This gap in the literature is a direction for future research.

3.2 Generalized Convex Clustering

3.2.1 Motivation

As has been discussed, convex clustering is an attractive method as it can be viewed as a convex relaxation of two well-studied and widely-used clustering methods, k -means and hierarchical clustering, while overcoming some of the pitfalls of those methods. In doing so, convex clustering (3.2) uses the squared error loss function when estimating the cluster centroids. However, in some situations it may make more sense to use a different loss function, such as a negative log-likelihood, as is done in generalized linear models (McCullagh & Nelder, 1989; Boos & Stefanski, 2013). For example, the “bag-of-words” model in document classification and natural language processing represents each document as a vector where the values correspond to the frequency that each word

appears in the document (Manning et al., 2008). Another common example comes from RNA sequencing data where each data point is the number of reads for a given sample and region of interest (Witten, 2011). In both situations, since the data to be clustered are non-negative counts, then either the negative-binomial or Poisson distribution would be a natural choice (Li & Zha, 2006; Witten, 2011; Wu et al., 2013). Other examples include the use of the beta distribution for clustering proportions such as array-based DNA methylation data (Houseman et al., 2008; Koestler et al., 2013; Ma et al., 2014), the Bernoulli distribution for clustering binary data (Li, 2006; Bouguila, 2010; Tamhane et al., 2010; Grantham, 2014) such as binary document classification (Li & Zhu, 2005), or the multinomial distribution for categorical data (Bontemps et al., 2013; Hasnat et al., 2017) such as movie ratings (Zhou & Lange, 2009).

3.2.2 Formulation

To this end, we propose the following generalized convex clustering model,

$$\sum_{i=1}^n \sum_{j=1}^p \ell(x_{ij}, u_{ij}) + \rho \sum_{i < j} \omega_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad (3.4)$$

where $\ell(x_{ij}, u_{ij})$ is a negative log-likelihood function, and ρ and ω_{ij} are defined as in (3.2). The choice of $\ell(x_{ij}, u_{ij})$ depends on the type of data to be clustered as mentioned in Section 3.2.1. For example, the Bernoulli model can be used to model binary data, with the log-likelihood given by

$$\mathcal{L}(\mathbf{p}, \mathbf{X}) = \sum_{i=1}^n \sum_{j=1}^p [x_{ij} \ln(p_{ij}) + (1 - x_{ij}) \ln(1 - p_{ij})], \quad (3.5)$$

where $p_{ij} = P(x_{ij} = 1) \in [0, 1]$ is the Bernoulli probability of success (Casella & Berger, 2001). For the canonical logit link function, we have

$$\text{logit}(p_{ij}) = \log\left(\frac{p_{ij}}{1 - p_{ij}}\right) = \theta_{ij} \Rightarrow p_{ij} = \frac{\exp(\theta_{ij})}{1 + \exp(\theta_{ij})}. \quad (3.6)$$

Plugging this expression for p_{ij} (3.6) into (3.5) and negating it, the loss function becomes

$$\ell(\Theta, \mathbf{X}) = \sum_{i=1}^n \sum_{j=1}^p [\ln(1 + \exp(\theta_{ij})) - \theta_{ij}x_{ij}].$$

Similarly, to model count data with the Poisson distribution, the log-likelihood function is

$$\mathcal{L}(\lambda, \mathbf{X}) = \sum_{i=1}^n \sum_{j=1}^p [x_{ij} \ln(\lambda_{ij}) - \lambda_{ij} - \ln(x_{ij}!)],$$

where $\lambda_{ij} > 0$ is the Poisson intensity parameter (Casella & Berger, 2001). Using the canonical log link function,

$$\log(\lambda_{ij}) = \theta_{ij} \Rightarrow \lambda_{ij} = \exp(\theta_{ij}),$$

the loss function is

$$\ell(\Theta, \mathbf{X}) = \sum_{i=1}^n \sum_{j=1}^p [\exp(\theta_{ij}) - x_{ij}\theta_{ij}].$$

3.2.3 Results

Similar to Witten (2011), our method was motivated by the desire to cluster an RNA sequencing dataset, dealing with breast cancer in our case (Allen & Liu, 2013). For this dataset, the true clusters are known, making it possible to calculate performance metrics, but generalized convex clustering (3.4) had very similar performance to convex clustering (3.2). As a result, we extensively studied the generalized formulation through numerous simulations, including the use of the Bernoulli, Poisson, Zero-inflated Poisson, and Multinomial distributions to generate the data. We also applied it to a handful of other datasets, including a document classification dataset, a dataset from the General Social Survey used by Lange (2016), and the soybean and mushroom datasets used in the clustering literature (Huang, 1998; Mampaey & Vreeken, 2013; Amiri et al., in press) and available from the UCI Machine Learning Repository (Lichman, 2013). However, in all of these settings the performance between the two methods was similar and the generalized formulation did not lead to a noteworthy improvement in performance, and thus the results are omitted.

3.3 Discussion

Although generalized convex clustering is a natural and intuitive extension, it did not result in better clustering. One possible explanation is that this similar performance could be further evidence of the importance that the weights play in the quality of the solution, since both generalized convex clustering and convex clustering used the same weights (3.3). For this reason, we also considered using model-based weights that are calculated using the likelihood function, but this also did not lead to a significant improvement over the original model. Another alternative was to incorporate an offset term in the model.

In Poisson models, an offset term is sometimes used when different observations have different levels of exposure to the event of interest. Effectively, this converts the data from counts to rates. However, this modification also did not impact the results. Thus, based on our results convex clustering is a situation where the squared error loss function was robust to different data generating processes, and the more-complex model was not beneficial.

Chapter 4

Convex Triclustering

4.1 Introduction

As discussed in Chapter 3, clustering attempts to identify groups of objects in a dataset. For a data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ with n observations (rows) and p variables (columns), clustering corresponds to assigning the rows of the data matrix to similar groups. Biclustering, or co-clustering, is the extension of clustering to two dimensions where both the observations (rows) and the variables or features (columns) of the data matrix \mathbf{X} are simultaneously grouped together. For example, in cancer research, one goal is to not only identify similar groups of cancerous tumors but also the genes that describe those tumor subgroups (Chi et al., 2017). In text mining, biclustering can be used to identify groups of documents that have similarities with respect to subgroups of words (Dhillon, 2001).

Triclustering is then the natural extension of co-clustering to three dimensions where the data are organized as a three-dimensional array, or *tensor*. With the ever-increasing availability and complexity of data, multidimensional data are progressively becoming

more common. One obvious example of the third dimension is time, where triclustering can uncover how biclustering relationships evolve over time. In fact, the most common application of triclustering in the literature is to gene expression data where gene expression levels are available across time under different conditions, such as different samples, experimental conditions, or individuals (Cheng & Church, 2000; Zhao & Zaki, 2005). Other tensor clustering examples involving time include email communications (sender, recipient, time) (Papalexakis et al., 2013), online chatroom communications (user, keyword, time) (Acar et al., 2006), bike rentals (source station, destination station, time) (Guigourès et al., 2015), and internet network traffic (source IP, destination IP, time) (Sun et al., 2006). There have also been tensor clustering applications that do not deal with time, such as bibliographic citations (author, conference, keyword) (Sun et al., 2006), movie ratings (users, movies, ratings) (Papalexakis et al., 2013), airline traffic (source airport, destination airport, airline) (Wu et al., 2016), and internet hyperlinks (source website, destination website, anchor text) (Kolda et al., 2005) to name a few.

In this chapter we propose *convex triclustering*, a convex formulation of the triclustering problem, and develop an algorithm for solving it. By casting the triclustering method as a convex optimization problem, the resulting solution is the unique global minimum and thus does not depend on the initialization. Additionally, this minimizer is continuous in the data, resulting in a stable solution in the face of small changes in the data. On the other hand, many of the existing approaches to clustering three-dimensional data, especially triclustering methods, rely on greedy search heuristic algorithms that are often unstable with respect to small changes in the data and different initializations, and are prone to return suboptimal solutions. Convex triclustering produces an entire solution path of triclusters as a function of only one tuning parameter, which alleviates the need to specify the number of triclusters beforehand. The solutions are also continuous

in this tuning parameter, so convex triclustering performs continuous clustering as the lasso (1.5) performs continuous variable selection. We develop automatic, data-driven methods for both selecting the tuning parameter and constructing the weights to make the method practically simple to use. Moreover, a MATLAB implementation of convex triclustering will be available in a forthcoming toolbox.

The rest of the chapter is organized as follows. The requisite background on tensors is presented in Section 4.2 before the tensor clustering literature is reviewed in Section 4.3. In Section 4.4, we present our convex formulation of triclustering while also discussing some of its key properties as well as our approach to solving the optimization problem. Some practical considerations for using convex triclustering are discussed in Section 4.5, and then the method is thoroughly studied through several simulations in Section 4.6. Our method is used to analyze a real online advertising dataset in Section 4.7 before Section 4.8 concludes.

4.2 Preliminaries: Tensor Background and Notation

4.2.1 Tensor Basics

Before proceeding further, we first define notation and provide the necessary background on tensors required for this chapter since tensors are not yet a standard part of a statistician’s toolkit as we believe they should be. For additional background information on tensors and their applications in statistics and machine learning, readers are referred to the excellent survey article by Kolda & Bader (2009) as well as more-recent overview articles that are geared more towards applications in data mining, signal processing, and machine learning (Cichocki et al., 2015; Papalexakis et al., 2016; Sidiropoulos et al., 2016). Given the popularity and usefulness of the Kolda & Bader (2009) overview paper,

our notation and definitions closely align with their conventions. One notable exception is that we use m to index to mode of the tensor (defined shortly) instead of n .

As before, a vector is denoted by a boldface lowercase letter, \mathbf{x} , and a matrix by a boldface capital letter, \mathbf{X} . A scalar is denoted by a lowercase letter, x , and is used with subscripts to represent an element in a vector or matrix. That is, x_i is the i th entry of the vector \mathbf{x} , while x_{ij} is located in row i and column j of the matrix \mathbf{X} . A tensor is a multidimensional array that generalizes vectors and matrices to more than two dimensions. In fact, a matrix is a second-order tensor and a vector, which itself is a one-dimensional matrix, is a first-order tensor. Here, the *order* of a tensor refers to its number of *modes* or dimensions, sometimes also called ways (Kolda & Bader, 2009). A general M th-order tensor, where $M \geq 3$, is denoted by a boldface Euler script letter, $\mathfrak{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_M}$, where $I_m \in \mathbb{N}$ is the length or size of mode m for $m = 1, 2, \dots, M$. A tensor is called *cubical* if every mode is the same size (Kolda & Bader, 2009). Henceforth we restrict our attention to third-order tensors, $\mathfrak{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, as that is the focus of this chapter.

Naturally, the (i, j, k) element of the third-order tensor \mathfrak{X} is denoted by x_{ijk} . We use MATLAB-style colon notation in a subscript to index all elements of a mode. For example, $\mathbf{x}_{:j}$ represents the entire j th column of the matrix \mathbf{X} . A *fiber* is the higher-order analogue of a row or column in a matrix. Specifically, a mode- m fiber is a vector obtained by fixing all of the indices except the m th index. A mode-1 (mode-2) fiber is synonymous to a matrix column (row). Fibers along the third dimension are called *tubes*, so a third-order tensor has column (mode-1), row (mode-2), and tube (mode 3) fibers, denoted by $\mathbf{x}_{:jk}$, $\mathbf{x}_{i:k}$, and $\mathbf{x}_{ij\cdot}$. A visual representation of the fibers in a third-order tensor are given in Figure 4.1. By convention, fibers are oriented as column vectors when extracted from a tensor (Kolda & Bader, 2009). In addition to extracting vectors from a tensor, we

can also extract matrices along each mode of a tensor by fixing only one index. These matrices are called *slices*, and a third-order tensor \mathbf{X} has horizontal ($\mathbf{X}_{i::}$), lateral ($\mathbf{X}_{:j:}$), and frontal ($\mathbf{X}_{::k}$) slices, as shown in Figure 4.2. Since the k th frontal slice corresponds to a standard matrix, it is also denoted more succinctly as \mathbf{X}_k (Kolda & Bader, 2009).

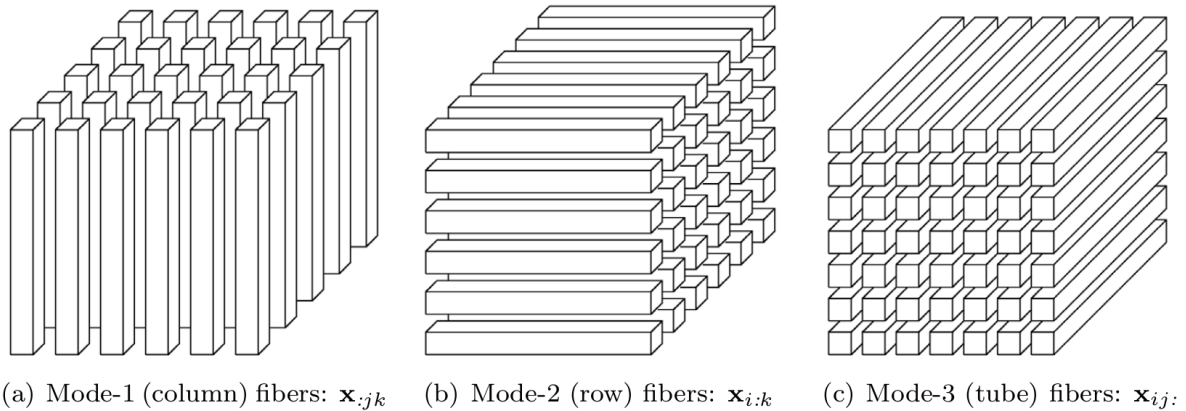


Figure 4.1: Fibers of a Third-order Tensor. Source: Kolda & Bader (2009).

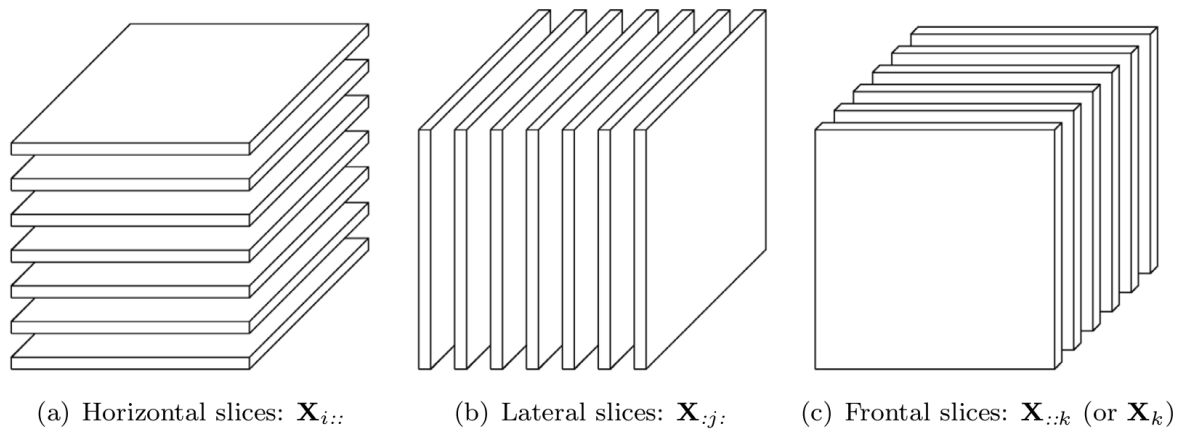


Figure 4.2: Slices of a Third-order Tensor. Source: Kolda & Bader (2009).

4.2.2 Tensor Operations

Now that the basics have been established, we will introduce the relevant tensor operations. The *flattening* of a tensor, also known as the *unfolding* or *matricization*, is a higher-order analog of vectorization where a tensor is converted to a matrix. For a matrix $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$, recall that the vectorization, $\text{vec}(\mathbf{X})$, involves sequentially stacking the columns of \mathbf{X} on top of each other to convert the matrix to a vector,

$$\text{vec}(\mathbf{X}) = \begin{pmatrix} \mathbf{x}_{:1} \\ \mathbf{x}_{:2} \\ \vdots \\ \mathbf{x}_{:I_2} \end{pmatrix}, \quad (4.1)$$

where $\text{vec}(\mathbf{X}) \in \mathbb{R}^{I_1 I_2}$. The mode- m flattening of a tensor \mathfrak{X} , then, is the matrix $\mathbf{X}_{(m)}$ whose columns are the mode- m fibers of the tensor. For a third-order tensor $\mathfrak{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, the mode-1 flattening is the matrix $\mathbf{X}_{(1)} \in \mathbb{R}^{I_1 \times (I_2 I_3)}$, for example. Vectorization can also be applied to a tensor, $\text{vec}(\mathfrak{X})$, in which the frontal slices \mathbf{X}_k are vectorized sequentially into one long vector,

$$\text{vec}(\mathfrak{X}) = \begin{pmatrix} \text{vec}(\mathbf{X}_{::1}) \\ \text{vec}(\mathbf{X}_{::2}) \\ \vdots \\ \text{vec}(\mathbf{X}_{::I_3}) \end{pmatrix} = \begin{pmatrix} \mathbf{x}_{:11} \\ \mathbf{x}_{:21} \\ \vdots \\ \mathbf{x}_{:I_2 1} \\ \mathbf{x}_{:12} \\ \vdots \\ \mathbf{x}_{:I_2 I_3} \end{pmatrix}. \quad (4.2)$$

The *norm* of a third-order tensor is the intuitive extension of the ℓ_2 vector norm and the Frobenius matrix norm to a tensor,

$$\|\mathbf{X}\| = \sqrt{\sum_{i,j,k} x_{ijk}^2}. \quad (4.3)$$

Recall that the outer product of two vectors $\mathbf{a} \in \mathbb{R}^{I_1}$ and $\mathbf{b} \in \mathbb{R}^{I_2}$, denoted by $\mathbf{a} \circ \mathbf{b}$, is a matrix $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ whose elements are the product of the corresponding vector elements, $x_{ij} = a_i b_j$. The third-order outer product is defined similarly as

$$\mathbf{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}, \quad (4.4)$$

for vectors $\mathbf{a} \in \mathbb{R}^{I_1}$, $\mathbf{b} \in \mathbb{R}^{I_2}$, and $\mathbf{c} \in \mathbb{R}^{I_3}$, where the tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ has entries $x_{ijk} = a_i b_j c_k$. Such a tensor is called *rank one*, and this construction of a tensor is illustrated in Figure 4.3. It is also possible to multiply a tensor by a matrix along a given mode. For a tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_m}$, the mode- m (or m -mode) matrix product, denoted $\mathbf{X} \times_m \mathbf{U}$, is of size $I_1 \times \cdots \times I_{m-1} \times J \times I_{m+1} \times \cdots \times I_M$ and is given by

$$(\mathbf{X} \times_m \mathbf{U})_{i_1 \cdots i_{m-1} j i_{m+1} \cdots i_M} = \sum_{i_m=1}^{I_m} x_{i_1 i_2 \cdots i_M} u_{j i_m}, \quad (4.5)$$

elementwise, so each mode- m fiber is multiplied by the matrix \mathbf{U} . Multiple mode- m products can be performed in any order,

$$\mathbf{X} \times_m \mathbf{A} \times_{\tilde{m}} \mathbf{B} = \mathbf{X} \times_{\tilde{m}} \mathbf{B} \times_m \mathbf{A}, \quad (m \neq \tilde{m}). \quad (4.6)$$

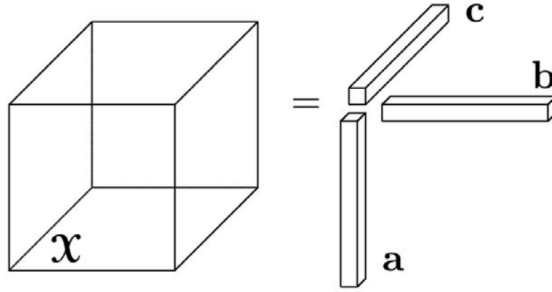


Figure 4.3: Rank-one Third-order Tensor. $\mathbf{X} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c}$, where the (i, j, k) element is $x_{ijk} = a_i b_j c_k$. Source: Kolda & Bader (2009).

4.2.3 Tensor Decompositions

To finish our basic overview of tensors, we introduce tensor decompositions. As with matrices, decompositions or factorizations play a vital role when using tensors for data analysis, and are arguably more crucial for tensors given the increased complexity of tensor data. There are two workhorse tensor decompositions, although it can often seem like more because each decomposition has multiple names, which sometimes refer to slight variations of one of the two main decompositions. Furthermore, one could argue that there is really only one primary tensor decomposition, as the one is a special case of the other.

We first introduce the CANDECOMP/PARAFAC decomposition (CPD), whose name is an artifact of its simultaneous rediscovery under the names CANDECOMP, for canonical decomposition (Carroll & Chang, 1970), and PARAFAC for parallel factors (Harshman, 1970; Kolda & Bader, 2009). The CPD factorizes a tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ into a linear combination of R rank-one component tensors,

$$\mathbf{X} \approx \sum_{r=1}^R \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r, \quad (4.7)$$

where $\mathbf{a}_r \in \mathbb{R}^{I_1}$, $\mathbf{b}_r \in \mathbb{R}^{I_2}$, $\mathbf{c}_r \in \mathbb{R}^{I_3}$, and R is a positive integer. A visual representation of the CPD is given in Figure 4.4. The smallest R such that (4.7) holds with equality is the *rank* of the tensor \mathfrak{X} (Kolda & Bader, 2009; Cichocki et al., 2015), and is sometimes called the CP rank to distinguish it from other definitions of tensor rank (Yokota et al., 2017). While so far most of the tensor concepts presented have been fairly straightforward generalizations of matrix concepts, that is not true for the rank of a tensor, as determining the rank of a tensor is NP-hard (Kolda & Bader, 2009). For the CPD of a third-order tensor (4.7), the three factor or component matrices are constructed by combining the component vectors along a given mode, as in

$$\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \cdots \ \mathbf{a}_R] \in \mathbb{R}^{I_1 \times R}, \quad (4.8)$$

and similarly for \mathbf{B} and \mathbf{C} . Using the factor matrices, it is common to re-express the CPD (4.7) more succinctly as $\mathfrak{X} = [[\mathbf{A}, \mathbf{B}, \mathbf{C}]]$ (Kolda & Bader, 2009). One can interpret the factor matrices as containing the latent, or hidden, factors that capture the underlying structure of the data along each mode of the tensor (Papalexakis et al., 2016). Unlike most matrix decompositions, under mild conditions the CPD is unique up to scaling and permuting the columns of the factor matrices. This property makes the CPD an attractive approach when interpretation is a goal of the analysis (Papalexakis et al., 2016).

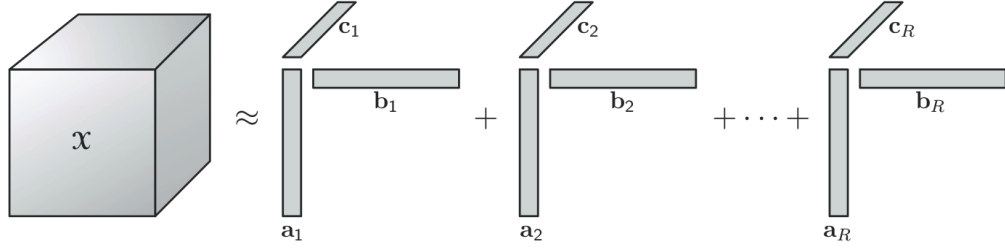


Figure 4.4: CP Decomposition. The CP decomposition (4.7) of a third-order tensor. Source: Kolda & Bader (2009).

The other main tensor decomposition is the Tucker decomposition, which was first proposed for third-order tensors by its namesake Tucker (1966) under the name of three-mode factor analysis (3MFA) or the Tucker3 model (Kolda & Bader, 2009). The Tucker decomposition, like the CPD, factors a third-order tensor \mathbf{X} into a set of three factor (component) matrices. The Tucker decomposition differs from the CPD in that its factorization also includes a core tensor \mathcal{G} that represents the interaction between the different components along each of the modes. Additionally, unlike the CPD, the Tucker decomposition does not require the number of factors along each mode to be the same. Formally, for a tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, its Tucker decomposition (Figure 4.5) is given by

$$\mathbf{X} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} = [[\mathcal{G}; \mathbf{A}, \mathbf{B}, \mathbf{C}]], \quad (4.9)$$

where $\mathbf{A} \in \mathbb{R}^{I_1 \times R_1}$, $\mathbf{B} \in \mathbb{R}^{I_2 \times R_2}$, and $\mathbf{C} \in \mathbb{R}^{I_3 \times R_3}$ are factor matrices along each mode, $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ is the core tensor representing a compressed version of \mathbf{X} , and $R_m \in \mathbb{Z}^+$ is the rank of the decomposition along mode m (Kolda & Bader, 2009). Typically the factor matrices are constrained to be orthogonal, in which case the Tucker decomposition is often referred to as the higher-order singular value decomposition (HOSVD) or the multilinear SVD (MLSVD) (De Lathauwer et al., 2000). The CPD is actually a special

case of the Tucker decomposition in which the core tensor is constrained to be cubical and *superdiagonal*, meaning $x_{ijk} \neq 0$ only if $i = j = k$. Unlike the CPD, the Tucker decomposition is not unique, but it is more flexible in that the rank of the decomposition does not have to be the same for every mode.

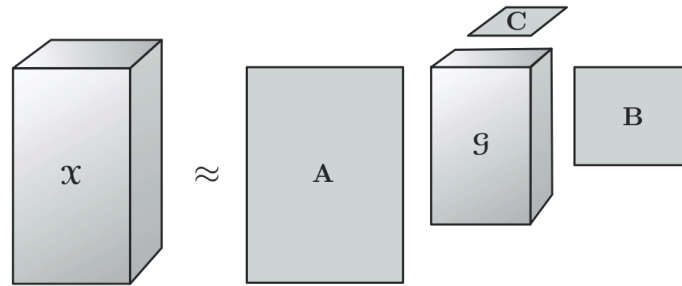


Figure 4.5: Tucker Decomposition. The Tucker decomposition (4.9) of a third-order tensor. Source: Kolda & Bader (2009).

4.3 Literature Review

4.3.1 Triclustering

In this section we review the literature on clustering tensor data. The clustering literature as it pertains to tensors can itself generally be classified into two or three main groups. Triclustering or tensor co-clustering is the area of the tensor clustering literature that is most relevant to our method. As mentioned, the goal of biclustering is to recover groups of rows and columns in a matrix that are similar to one another. When the rows and columns are reordered according to their groupings, a “checkerboard” pattern emerges (Chi et al., 2017). The goal of triclustering is then the natural extension of this idea to three dimensions. Now, the aim is to identify groups of rows, columns, and tubes that are similar to one another. A “checkbox” pattern appears when the fibers

along each mode are ordered according to their cluster assignments (Figure 4.6). These non-overlapping subtensors correspond to the different triclusters embedded in the larger data tensor.

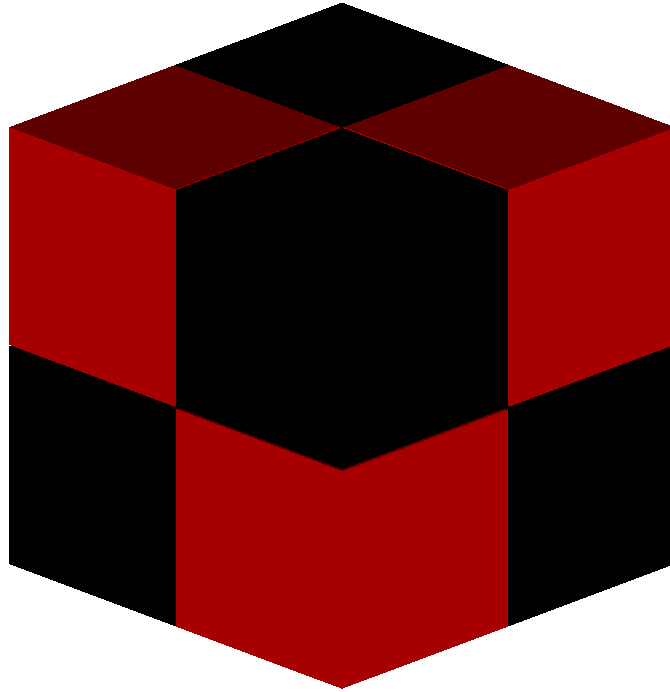


Figure 4.6: Tensor with Checkbox Structure. Each mode has two clusters for a total of eight triclusters.

The majority of methods in this strand of the tensor clustering literature have been motivated by the application of triclustering to gene expression data. Gene microarray data contain gene expression levels under different conditions, such as different samples, experimental conditions, or individuals (Cheng & Church, 2000). A common application of biclustering, there has also been interest in analyzing gene expressions across time (Bar-

Joseph, 2004; Conesa et al., 2006) where the data naturally fit into a third-order tensor. This has spawned the development of several triclustering methods designed specifically for gene expression data where time is included as a third dimension, the first of which was **Tricluster** proposed by Zhao & Zaki (2005) with a parallel implementation of the algorithm (**ParTricluster**) later developed by Braga Araújo et al. (2008). **Tricluster** first applies a graph-based approach to each gene-by-condition frontal slice to identify biclusters at each time point. Then, this process is repeated with the biclusters from the different time points serving as the nodes in the graph to identify the final triclusters.

Although a seminal work, several triclustering methods have since been proposed to address some of the shortcomings of **Tricluster**. Jiang et al. (2006) proposed **gTricluster**, which extended and generalized **Tricluster** to make it more flexible and robust to noise, mainly by using the Spearman rank correlation to measure tricluster similarity. The Order Preserving Tricluster (**OPTricluster**) method is a related approach that is specifically aimed at short time series where there are only 3-8 observations along the time dimension (Tchagang et al., 2012). Both **LagMiner** (Xu et al., 2009) and **td-cluster** (Wang et al., 2010) were developed to handle time-delayed patterns, as one gene's relationship with another gene may occur downstream in a pathway (Yu et al., 2003). Gutiérrez-Avilés et al. (2014) took a different approach and proposed a genetic algorithm for the sake of finding gene triclusters. More recently, Kakati et al. (2016) took an approach similar in spirit to **Tricluster** but adopted a similarity measure from the gene biclustering literature (Ahmed et al., 2014) that is based on the local mean of the genes for each condition, again to improve the types of patterns that can be identified by the algorithm.

Another strand of the triclustering research for gene data has focused on extending a classic gene expression biclustering method to three dimensions. One of the first works to

develop a biclustering method for gene data analysis was by Cheng & Church (2000). The goal was to find large biclusters with a small mean squared residue (MSR) score, which is a measure of the variability or homogeneity within a bicluster. The biclusters are found using a greedy search heuristic by removing rows and columns that reduce MSR until MSR is below a user-specified threshold. Both `TriWClustering` (Dede & Oğul, 2013) and δ -`Trimax` (Bhar et al., 2013) are seemingly independent attempts at developing a tricluster version of the Cheng & Church (2000) algorithm, while Gutiérrez-Avilés & Rubio-Escudero (2014) simply focused on developing a three-dimensional version of mean squared residue. Tang et al. (2017) was not motivated by gene data but also came up with a greedy triclustering approach that is very similar to `TriWClustering` and δ -`Trimax`. An extension of δ -`Trimax`, called `EMOA- δ -Trimax`, has also been developed to identify overlapping triclusters (Bhar et al., 2015).

Although the majority of the work in this area of the triclustering literature was specifically motivated by gene data, there have also been developments aimed at other applications. As we will see, tensor decompositions play a crucial role in many approaches to clustering tensor data. One such approach to triclustering is a non-negative CPD with sparsity imposed on the latent factors via lasso-type penalties as developed by Papalexakis et al. (2013). By imposing sparsity and non-negativity on the factors, they can then be interpreted as the soft (fuzzy) clustering weights along each mode. Sun et al. (2009) performed a Tucker decomposition on the data tensor and then applied k -means to the factor matrices to obtain the clusters along each mode, which are then combined to find the triclusters.

A few other triclustering methods that do not utilize tensor decompositions are of note. Guigourès et al. (2015) used a graph-based approach in lieu of tensor machinery for finding triclusters when one mode represents time. An adjacency matrix for each time

point is biclustered, and then the time dimension is clustered by looking for stationary edge distributions across time. Liu et al. (2015) extended fuzzy c-means (Dunn, 1973; Bezdek, 1981), which is essentially the soft clustering version of k -means, to handle three-dimensional data. Wu et al. (2016) proposed a higher-order extension of spectral clustering using higher-order random walks, which is a promising idea but is limited in its current form as it is developed only for tensors that are non-negative, square, and symmetric.

4.3.2 Multi-way Clustering

Multi-way clustering, also referred to as single-mode (Jegelka et al., 2009) or multi-view (Zhao et al., 2017) clustering, is another segment of the tensor clustering literature where the interest is in clustering along only one mode of the tensor. For example, Acar et al. (2006) analyzed chatroom communications where the third-order tensor consisted of (users, keywords, time), but the interest was in clustering along only one dimension (users). Similarly, multi-view learning is an area of machine learning research that aims to incorporate different “views” by combining data from multiple sources (Zhao et al., 2017). Thus, like traditional clustering, the goal is to cluster the rows but single-mode tensor clustering incorporates information from other dimensions as well.

A few tensor-based approaches have been proposed, often involving tensor decompositions. Acar et al. (2006) performed a Tucker decomposition of the data tensor, and then applied k -means to the mode-1 factor matrix to cluster the users. A similar decomposition plus k -means approach is followed by Kutty et al. (2011), except a new Tucker-like decomposition is proposed that is able to handle large, sparse tensors. Instead of performing the Tucker decomposition and k -means sequentially, Vichi et al. (2007) formulated the problem to perform simultaneous clustering along mode 1 and dimension reduction

along modes 2 and 3. Instead of working with the original data tensor, Liu et al. (2013) first created a similarity (affinity) matrix for each data source using a Gaussian kernel, similar in spirit to spectral clustering. The similarity matrices are stacked together into a similarity tensor, which is decomposed using the Tucker decomposition before k -means is applied to the mode-1 factor matrix as before. An alternative approach that does not involve a tensor decomposition was proposed by Hařan et al. (2008). It involves sampling horizontal slices from the original tensor and then clustering the columns, so the resulting cluster centroids can be used to quickly construct a smoothed estimate of the original tensor for the sake of quickly rendering images.

4.3.3 Clustering Tensor Objects

Another form of tensor clustering in the literature is a setting in which the objects to be clustered are themselves tensors. If the tensors of interest are second-order tensors (matrices), then this line of the research is fundamentally similar to the multi-way tensor clustering research (Section 4.3.2). Technically, the traditional clustering of n data points in \mathbb{R}^p falls into this category since vectors are first-order tensors, so this type of tensor clustering is another natural extension of clustering to higher-order data. A common motivating example for this type of tensor clustering involves image data. Images are often represented by a matrix where the values in the matrix correspond to the raw pixels or features extracted from the image (Cao et al., 2015). One approach to clustering images is to first vectorize each image matrix and then use a standard clustering algorithm on a matrix where each observation/row is a vectorized image. While this approach is effective, maintaining the images as matrices can preserve the spatial relationships between the pixels of the images (Cao et al., 2015; He et al., 2005).

To this end, there have been several approaches developed with the aim of clustering

images while maintaining their natural matrix form, and once again tensor decompositions play a starring role. He et al. (2005) attempted to find an optimal two-dimensional subspace representation of the tensor, and then a classic clustering algorithm such as k -means can be applied to obtain a final clustering result. Huang et al. (2008) established a connection between the HOSVD (Section 4.2.3) and k -means clustering. They show that performing an HOSVD and applying k -means clustering to the mode-3 factor matrix (corresponding to the observation index), is equivalent to jointly factorizing the set of image matrices using the two-dimensional singular value decomposition (2DSVD) proposed by Ding & Ye (2005) and then using a matrix-valued k -means clustering on the lower-dimensional matrices resulting from the 2DSVD. This result provides theoretical justification for the common approach of using k -means clustering on the factor matrices obtained from a CP or Tucker decomposition, as discussed earlier. Other attempts have been made to modify the decomposition so that it simultaneously performs dimension reduction and clustering similar to Vichi et al. (2007), instead of performing the tasks sequentially. With this in mind, Peng & Li (2011), Zhang et al. (2013), and Sun et al. (2016) all made modifications to the HOSVD by imposing different constraints on the factors. For the observation index mode, Peng & Li (2011) constrained the factor matrix to be binary with only one non-zero entry per row to represent the cluster assignments, while Sun et al. (2016) imposed simplex constraints so that the factors serve as fuzzy clustering weights. Alternatively, Zhang et al. (2013) developed the **Tri-ONTD** (tri-factor orthogonal non-negative tensor decomposition) method that is similar to the HOSVD but imposes non-negativity constraints on all of the factor matrices. Non-negative matrix factorization has been used for biclustering (Ding et al., 2006), and this model is the extension of that to three-way data. Lastly, Metzler & Miettinen (2015) used a decomposition designed specifically for binary data, such as for clustering a sequence of adjacency

matrices. Like Peng & Li (2011), Metzler & Miettinen (2015) then used the constrained factor matrices to determine the cluster assignments.

Most of the earlier works in this strand of tensor clustering were motivated by clustering matrices, but efforts have also been made to extend this idea to clustering objects that are tensors of higher-order. Cao et al. (2015) first created a low-rank approximation of each order- M tensor to be clustered. These approximations are constructed by using a CPD-like decomposition except the ℓ_1 norm is used to make the decomposition more robust to noise. The approximated tensors are stacked together into a tensor of order $M + 1$, denoted \mathcal{Z} , and the HOSVD of \mathcal{Z} is computed. The resulting factor matrix along mode $M + 1$, corresponding to the observation index, is then clustered with k -means to obtain the final clustering result. The aforementioned decomposition developed by Sun et al. (2016), which imposes simplex constraints on one of factor matrices and is referred to as the heterogeneous Tucker decomposition, was also designed to cluster tensors of order M by first arranging them in a tensor of order $M + 1$ before applying the decomposition. The idea behind Sun et al. (2014) is to perform a Tucker decomposition on each tensor separately, and then apply k -means on the resulting (vectorized) core tensors, but their method performs these steps simultaneously. Ultimately, clustering tensor objects is not the focus of our immediate work, but is included in the literature review to give a more complete view of the existing tensor clustering literature. The possibility of extending our method to higher-order tensors is left as future work.

4.4 A Convex Formulation of Triclustering

4.4.1 Formulation

To identify triclusters embedded in a third-order data tensor, we propose minimizing the convex optimization problem given by

$$F_\rho(\mathbf{u}) = \frac{1}{2}\|\mathbf{x} - \mathbf{u}\|_F^2 + \rho[J_1(\mathbf{u}) + J_2(\mathbf{u}) + J_3(\mathbf{u})], \quad (4.10)$$

where

$$\begin{aligned} J_1(\mathbf{u}) &= \sum_{i < j} \omega_{1,ij} \|\mathbf{u}_{i::} - \mathbf{u}_{j::}\|_F, \\ J_2(\mathbf{u}) &= \sum_{i < j} \omega_{2,ij} \|\mathbf{u}_{:i} - \mathbf{u}_{:j}\|_F, \\ J_3(\mathbf{u}) &= \sum_{i < j} \omega_{3,ij} \|\mathbf{u}_{::i} - \mathbf{u}_{::j}\|_F, \end{aligned} \quad (4.11)$$

$\mathbf{x} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is the data tensor to be clustered, $\mathbf{u} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ is the tensor of tricluster means to be estimated, $\rho \geq 0$ is the penalization parameter, and $\omega_{m,ij}$ are non-negative weights for $m = 1, 2, 3$. The first term is the loss function that measures how well \mathbf{u} fits \mathbf{x} , while $J_m(\mathbf{u})$ is a penalty term that shrinks together the i th and j th mode- m slices of \mathbf{u} . By taking $J(\mathbf{u}) = J_1(\mathbf{u}) + J_2(\mathbf{u}) + J_3(\mathbf{u})$, we see that this model fits into the penalized estimation formulation discussed in Chapter 1. The penalty term has three parts, one for each mode of the third-order true mean tensor, which is a natural generalization of the convex biclustering penalty proposed by Chi et al. (2017). The convex biclustering formulation simultaneously penalizes the rows and columns of the data matrix, where here the equivalent is to simultaneously penalize the three types of

slices in the tensor (Figure 4.2). As a result, the parameter estimates are fused together in all directions, which encourages the estimated tensor to have the desired checkbox structure (Figure 4.6).

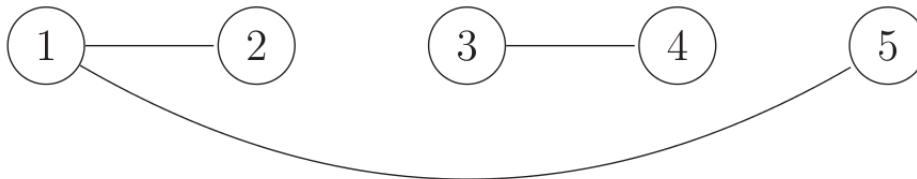


Figure 4.7: Example Weights-Induced Edge Graph. A graph with positive weights for $\omega_{m,12}$, $\omega_{m,15}$, and $\omega_{m,34}$ and zero weights between all other nodes, corresponding to the mode- m slices. Source: Chi & Lange (2015).

The pre-specified weights, $\omega_{m,ij} \geq 0$, can fine tune the shrinkage of the slices along mode m . For the m th mode, we can take a graphical point of view of the clustering and define the set \mathcal{E}_m as the edge set of the underlying graph (Chi & Lange, 2015). Each slice is a node in the graph and the set \mathcal{E}_m contains an edge (i, j) if and only if $\omega_{m,ij} > 0$. For example, consider the toy edge graph given in Figure 4.7 and assume without loss of generality the graph corresponds to mode 1. Given the connectivity of the graph, slices $\mathbf{u}_{1::}$, $\mathbf{u}_{2::}$, and $\mathbf{u}_{5::}$ will all be shrunk towards each other, as will slices $\mathbf{u}_{3::}$ and $\mathbf{u}_{4::}$. Since $\omega_{m,ij} = 0$ for any $(i, j) \notin \mathcal{E}_m$, then we can equivalently write the penalty terms for each

mode as

$$\begin{aligned}
J_1(\mathbf{u}) &= \sum_{(i,j) \in \mathcal{E}_1} \omega_{1,ij} \|\mathbf{u}_{i::} - \mathbf{u}_{j::}\|_F, \\
J_2(\mathbf{u}) &= \sum_{(i,j) \in \mathcal{E}_2} \omega_{2,ij} \|\mathbf{u}_{:i} - \mathbf{u}_{:j}\|_F, \\
J_3(\mathbf{u}) &= \sum_{(i,j) \in \mathcal{E}_3} \omega_{3,ij} \|\mathbf{u}_{::i} - \mathbf{u}_{::j}\|_F.
\end{aligned}$$

Further discussion of the weights and practical recommendations for constructing them will be discussed in Section 4.5.1 and Section 4.6.4.

The penalization parameter ρ governs the extent to which the parameter estimates are fused together. When $\rho = 0$, the penalty term drops out and the objective function (4.10) is minimized by $\hat{\mathbf{u}} = \mathbf{x}$, so every element is its own tricluster. As ρ increases, more emphasis is placed on the penalty terms which forces elements in the estimated tensor to fuse together, resulting in a smoothed estimate of the data tensor. This fusion plays an important role in the formulation as it determines the cluster assignments along each mode. For example, two slices along mode 1, $\mathbf{x}_{i::}$ and $\mathbf{x}_{j::}$, belong to the same mode-1 partition if $\hat{\mathbf{u}}_{i::} = \hat{\mathbf{u}}_{j::}$. The clustering assignments for modes 2 and 3 are done analogously. Eventually, for a large enough ρ , there is extensive fusion and the parameter estimates are reflective of the graph induced by the weights along each mode. When the underlying graphs are fully connected, then the completely-fused parameter estimates correspond to the grand mean,

$$\hat{\mathbf{u}} = \bar{\mathbf{x}} = \frac{1}{I_1 I_2 I_3} \sum_{ijk} x_{ijk},$$

and only one trivial tricluster is identified. For intermediate values of ρ , the single-mode

cluster assignments are used to determine the triclusters. This approach to determining the triclusters was theoretically justified by Jegelka et al. (2009). Thus this formulation produces an entire solution path of triclustering results, similar in spirit to hierarchical clustering, as a function of only one tuning parameter ρ . This is a nice feature of the framework as it alleviates the need to specify the number of clusters a priori, as is the case with some clustering methods such as k -means. A method for automatically choosing an optimal ρ to produce a final triclustering result will be discussed in Section 4.5.2.

4.4.2 Properties

The convex triclustering formulation has several nice properties, which we now explore. It is worth noting that these properties hold for any algorithm used to minimize (4.10), as they are intrinsic to its convex formulation. The convex formulation provides not only algorithmic flexibility but also leaves open the possibility of later using a new, more efficient algorithm to solve (4.10).

Proposition 1 *The function $F_\rho(\mathbf{U})$ defined in (4.10) has a unique global minimizer.*

Proof. To see this, note that if a function is continuous and coercive, then its minimal point exists (Lange, 2013). A function $f(\mathbf{x})$ is said to be coercive if

$$\lim_{\|\mathbf{x}\| \rightarrow \infty} f(\mathbf{x}) = \infty, \quad (4.12)$$

which holds for $F_\rho(\mathbf{U})$ for each ρ (Lange, 2013). Additionally, $F_\rho(\mathbf{U})$ is continuous as a function of \mathbf{U} for a fixed ρ , so it has a minimal point. Furthermore, since $F_\rho(\mathbf{U})$ is strictly convex, then its minimal point is the unique global minimizer (Lange, 2013). Recall that a function is strictly convex if the relation (3.1) holds with strict inequality. ■

The existence of a unique global minimizer is an attractive feature of convex triclustering since the solution will always be the optimal point. The triclustering assignment does not depend on the initialization, and there is no risk of being stuck in a local minimum.

Denote $\mathbf{W}_m = \{w_{m,ij}\}$ as the weights matrix for mode m .

Proposition 2 *The minimizer \mathbf{U}^* of (4.10) is jointly continuous in $(\mathbf{X}, \rho, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3)$.*

Proof. The proof follows directly from the one given by Chi et al. (2017) for convex biclustering, with $\mathbf{w} = (\text{vec}(\mathbf{W}_1)^T, \text{vec}(\mathbf{W}_2)^T, \text{vec}(\mathbf{W}_3)^T)^T$ and $J(\mathbf{U}) = J_1(\mathbf{U}) + J_2(\mathbf{U}) + J_3(\mathbf{U})$. ■

The first implication of Proposition 2 relates to the algorithm implementation for solving the convex triclustering problem (4.10). Since the optimal solution \mathbf{U}^* is a continuous function of the penalization parameter ρ , it encourages the use of warm starts when estimating the solution path. That is, the solution for one ρ can be used as the initial guess for a slightly larger or smaller ρ , since small changes in ρ will result in only small changes in \mathbf{U}^* . Empirically, the use of warm starts can lead to a non-trivial reduction in computation time (Chi & Lange, 2015). From the continuity in ρ we also see that convex triclustering performs continuous clustering just as the lasso (1.5) performs continuous variable selection. Continuity of \mathbf{U}^* in \mathbf{X} provides an attractive stability result. Since \mathbf{U}^* varies smoothly with the data, small perturbations in the data will not lead to large variability of \mathbf{U}^* , or large variability in the cluster assignments since the differences of slices along each mode are used for the single-mode clustering results.

4.4.3 Estimation

Now we turn our attention to solving the proposed convex triclustering problem (4.10). First, define $\Delta_{m,ij} = \mathbf{e}_i - \mathbf{e}_j$ where \mathbf{e}_i is the i th standard basis vector in \mathbb{R}^{I_m} . That is,

\mathbf{e}_i has a 1 in the i th element and 0 otherwise. For a general tensor $\mathfrak{Z} \in \mathbb{R}^{I_1 \times \dots \times I_M}$ and a matrix $\mathbf{A} \in \mathbb{R}^{L \times I_m}$, we have the identity

$$\text{vec}(\mathfrak{Z} \times_m \mathbf{A}) = (\mathbf{I}_{I_M} \otimes \dots \otimes \mathbf{I}_{I_{m+1}} \otimes \mathbf{A} \otimes \mathbf{I}_{I_{m-1}} \otimes \dots \otimes \mathbf{I}_{I_1}) \mathbf{z},$$

where $\mathbf{z} = \text{vec}(\mathfrak{Z})$ and \times_m denotes the mode- m product defined by equation 4.6 (Cichocki et al., 2015). Using this identity, we can rewrite the penalty term for mode m as

$$\begin{aligned} J_m(\mathbf{u}) &= \sum_{(i,j) \in \mathcal{E}_m} w_{m,ij} \|\mathbf{u} \times_m \Delta_{m,ij}^T\|_F \\ &= \sum_{(i,j) \in \mathcal{E}_m} w_{m,ij} \|\text{vec}(\mathbf{u} \times_m \Delta_{m,ij}^T)\|_2 \\ &= \sum_{(i,j) \in \mathcal{E}_m} w_{m,ij} \|\mathbf{A}_{m,ij} \mathbf{u}\|_2, \end{aligned}$$

where $\mathbf{A}_{m,ij} = (\mathbf{I}_{I_M} \otimes \dots \otimes \mathbf{I}_{I_{m+1}} \otimes \Delta_{m,ij}^T \otimes \mathbf{I}_{I_{m-1}} \otimes \dots \otimes \mathbf{I}_{I_1})$ and $\mathbf{u} = \text{vec}(\mathbf{U})$. We can rewrite the loss function by also vectorizing the data tensor,

$$\frac{1}{2} \|\mathfrak{X} - \mathbf{u}\|_F^2 = \frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2,$$

where $\mathbf{x} = \text{vec}(\mathfrak{X})$. To clean up the notation, we define $l = (i, j)$ to represent a pair of slices, so the objective function becomes

$$\frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \rho \sum_m \sum_{l \in \mathcal{E}_m} w_{m,l} \|\mathbf{A}_{m,l} \mathbf{u}\|_2. \quad (4.13)$$

Our approach to minimizing equation (4.13) is to use standard variable splitting by introducing the dummy variable $\mathbf{v}_{m,l} = \mathbf{A}_{m,l} \mathbf{u}$ inside of the norm in the penalty terms. Let \mathbf{V}_m denote the $\left(\frac{I_1 I_2 I_3}{I_m}\right)$ -by- $|\mathcal{E}_m|$ matrix whose l th column is $\mathbf{v}_{m,l}$. Additionally, let

$\mathbf{v}_m = \text{vec}(\mathbf{V}_m)$ and $\mathbf{v} = [\mathbf{v}_1^T, \mathbf{v}_2^T, \mathbf{v}_3^T]^T$. Thus, we are interested in solving a constrained minimization problem

$$\begin{aligned} & \underset{\mathbf{v}, \mathbf{u}}{\text{minimize}} && \frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \rho \sum_m \sum_{l \in \mathcal{E}_m} w_{m,l} \|\mathbf{v}_{m,l}\|_2 && (4.14) \\ & \text{subject to} && \mathbf{v}_m = \mathbf{A}_m \mathbf{u} \end{aligned}$$

where $\mathbf{A}_m = (\mathbf{I}_{I_M} \otimes \cdots \otimes \mathbf{I}_{I_{m+1}} \otimes \mathbf{\Phi}_m \otimes \mathbf{I}_{I_{m-1}} \otimes \cdots \otimes \mathbf{I}_{I_1})$ and $\mathbf{\Phi}_m$ is the oriented edge-vertex incidence matrix for the m th mode graph

$$\Phi_{m,lv} = \begin{cases} 1 & \text{If node } v \text{ is the head of edge } l \\ -1 & \text{If node } v \text{ is the tail of edge } l \\ 0 & \text{otherwise,} \end{cases}$$

for $m = 1, 2, 3$.

To this end, we introduce Lagrange multipliers (dual variables) $\boldsymbol{\lambda}_m$ for the equality constraints, and denote $\boldsymbol{\Lambda}_m$ as the $\left(\frac{I_1 I_2 I_3}{I_m}\right)$ -by- $|\mathcal{E}_m|$ matrix whose l th column is $\boldsymbol{\lambda}_{m,l}$. Additionally, let $\boldsymbol{\lambda}_m = \text{vec}(\boldsymbol{\Lambda}_m)$ and $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_1^T, \boldsymbol{\lambda}_2^T, \boldsymbol{\lambda}_3^T]^T$. The Lagrangian for the optimization problem (4.13) is then

$$\begin{aligned} \mathcal{L}(\mathbf{u}, \mathbf{v}, \boldsymbol{\lambda}) &= \frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \sum_m \sum_{l \in \mathcal{E}_m} (\rho w_{m,l} \|\mathbf{v}_{m,l}\|_2 + \langle \boldsymbol{\lambda}_{m,l}, \mathbf{A}_{m,l} \mathbf{u} - \mathbf{v}_{m,l} \rangle) && (4.15) \\ &= \left[\frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \sum_m \langle \mathbf{A}_m^T \boldsymbol{\lambda}_m, \mathbf{u} \rangle \right] + \left[\sum_m \sum_{l \in \mathcal{E}_m} (\rho w_{m,l} \|\mathbf{v}_{m,l}\|_2 - \langle \boldsymbol{\lambda}_{m,l}, \mathbf{v}_{m,l} \rangle) \right]. \end{aligned}$$

The Lagrange dual function is

$$\begin{aligned}
\mathcal{D}(\boldsymbol{\lambda}) &= \min_{\mathbf{u}, \mathbf{v}} \mathcal{L}(\mathbf{u}, \mathbf{v}, \boldsymbol{\lambda}) \\
&= \min_{\mathbf{u}, \mathbf{v}} \left\{ \left[\frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \sum_m \langle \mathbf{A}_m^T \boldsymbol{\lambda}_m, \mathbf{u} \rangle \right] + \right. \\
&\quad \left. \left[\sum_m \sum_{l \in \mathcal{E}_m} (\rho w_{m,l} \|\mathbf{v}_{m,l}\|_2 - \langle \boldsymbol{\lambda}_{m,l}, \mathbf{v}_{m,l} \rangle) \right] \right\} \\
&= \min_{\mathbf{u}} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \sum_m \langle \mathbf{A}_m^T \boldsymbol{\lambda}_m, \mathbf{u} \rangle \right] + \\
&\quad \min_{\mathbf{v}} \left[\sum_m \sum_{l \in \mathcal{E}_m} (\rho w_{m,l} \|\mathbf{v}_{m,l}\|_2 - \langle \boldsymbol{\lambda}_{m,l}, \mathbf{v}_{m,l} \rangle) \right].
\end{aligned}$$

We will minimize these two parts separately. We can rewrite the first term as

$$\min_{\mathbf{u}} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \left\langle \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m, \mathbf{u} \right\rangle \right], \quad (4.16)$$

since the inner product $\langle \cdot, \cdot \rangle$ satisfies the distributive law. To clean up notation, define $\mathbf{z} = \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m$, so the problem (4.16) becomes

$$\min_{\mathbf{u}} \left[\frac{1}{2} \|\mathbf{x} - \mathbf{u}\|_2^2 + \langle \mathbf{z}, \mathbf{u} \rangle \right]. \quad (4.17)$$

To find the minimum, we take the derivative of (4.17) with respect to \mathbf{u} and set it equal

to zero. This results in $\mathbf{u}^* = \mathbf{x} - \mathbf{z}$, which we plug back into (4.17) and simplify to get

$$\begin{aligned}
& \frac{1}{2} \|\mathbf{x} - (\mathbf{x} - \mathbf{z})\|_2^2 + \langle \mathbf{z}, \mathbf{x} - \mathbf{z} \rangle \\
&= \frac{1}{2} \|\mathbf{z}\|_2^2 - \|\mathbf{z}\|_2^2 + \langle \mathbf{z}, \mathbf{x} \rangle \\
&= -\frac{1}{2} \|\mathbf{z}\|_2^2 + \langle \mathbf{z}, \mathbf{x} \rangle - \frac{1}{2} \|\mathbf{x}\|_2^2 + \frac{1}{2} \|\mathbf{x}\|_2^2 \\
&= -\frac{1}{2} \|\mathbf{x} - \mathbf{z}\|_2^2 + \frac{1}{2} \|\mathbf{x}\|_2^2 \\
&= -\frac{1}{2} \|\mathbf{x} - \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m\|_2^2 + \frac{1}{2} \|\mathbf{x}\|_2^2, \tag{4.18}
\end{aligned}$$

for the first part of the dual function.

For the second part, recall that for a convex function $f : \mathbb{R}^p \rightarrow \mathbb{R}$, its conjugate function $f^* : \mathbb{R}^p \rightarrow \mathbb{R}$ is defined as

$$f^*(\mathbf{b}) = \sup_{\mathbf{a}} (\langle \mathbf{b}, \mathbf{a} \rangle - f(\mathbf{a})),$$

which is closed and convex (Boyd & Vandenberghe, 2004). For $f(\mathbf{a}) = \|\mathbf{a}\|_2$, we have

$$f^*(\mathbf{b}) = \sup_{\mathbf{a}} (\langle \mathbf{b}, \mathbf{a} \rangle - \|\mathbf{a}\|_2) = \begin{cases} 0, & \|\mathbf{b}\|_2 \leq 1 \\ \infty, & \text{otherwise,} \end{cases} \tag{4.19}$$

so $f^*(\mathbf{b})$ is the indicator function for the closed and convex set $\{\mathbf{b} : \|\mathbf{b}\|_2 \leq 1\}$. Also, the conjugate function of a separable sum is the sum of the conjugate functions,

$$f(\mathbf{a}_1, \mathbf{a}_2) = g(\mathbf{a}_1) + g(\mathbf{a}_2) \Rightarrow f^*(\mathbf{b}_1, \mathbf{b}_2) = g^*(\mathbf{b}_1) + g^*(\mathbf{b}_2). \tag{4.20}$$

Using these results, we can simplify the second part of the dual function $\mathcal{D}(\boldsymbol{\lambda})$ as

$$\begin{aligned}
& \min_{\mathbf{v}} \left[\sum_m \sum_{l \in \mathcal{E}_m} (\rho w_{m,l} \|\mathbf{v}_{m,l}\|_2 - \langle \boldsymbol{\lambda}_{m,l}, \mathbf{v}_{m,l} \rangle) \right] \\
&= -\max_{\mathbf{v}} \left[\sum_m \sum_{l \in \mathcal{E}_m} (\langle \boldsymbol{\lambda}_{m,l}, \mathbf{v}_{m,l} \rangle - \rho w_{m,l} \|\mathbf{v}_{m,l}\|_2) \right] \\
&= -\sum_m \sum_{l \in \mathcal{E}_m} \left[\max_{\mathbf{v}_l} (\langle \boldsymbol{\lambda}_{m,l}, \mathbf{v}_{m,l} \rangle - \rho w_{m,l} \|\mathbf{v}_{m,l}\|_2) \right] \\
&= -\sum_m \sum_{l \in \mathcal{E}_m} \left[\max_{\mathbf{v}_l} \left(\left\langle \frac{\boldsymbol{\lambda}_{m,l}}{\rho w_{m,l}}, \mathbf{v}_{m,l} \right\rangle - \|\mathbf{v}_{m,l}\|_2 \right) \right] \\
&= -\sum_m \sum_{l \in \mathcal{E}_m} \delta_{C_{m,l}}(\boldsymbol{\lambda}_{m,l}), \tag{4.21}
\end{aligned}$$

where $\delta_{C_{m,l}}$ is the indicator function of the set $C_{m,l} = \{\mathbf{b} : \|\mathbf{b}\|_2 \leq \rho w_{m,l}\}$. Thus putting (4.18) and (4.21) together, we have

$$\mathcal{D}(\boldsymbol{\lambda}) = -\frac{1}{2} \|\mathbf{x} - \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m\|_2^2 + \frac{1}{2} \|\mathbf{x}\|_2^2 - \sum_m \sum_{l \in \mathcal{E}_m} \delta_{C_{m,l}}(\boldsymbol{\lambda}_{m,l}). \tag{4.22}$$

The corresponding dual problem is to maximize $\mathcal{D}(\boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$, or equivalently we can minimize $-\mathcal{D}(\boldsymbol{\lambda})$. Thus, the convex triclustering dual problem is

$$\min_{\boldsymbol{\lambda}} \frac{1}{2} \|\mathbf{x} - \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m\|_2^2 + \sum_m \sum_{l \in \mathcal{E}_m} \delta_{C_{m,l}}(\boldsymbol{\lambda}_{m,l}), \tag{4.23}$$

and

$$\mathbf{u} = \mathbf{x} - \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m,$$

can be used to recover the primal solution. We solve the dual problem (4.23) using prox-

imal gradient descent, also known as forward-backward splitting (Combettes & Wajs, 2005; Combettes & Pesquet, 2011; Goldstein et al., 2014). The proximal gradient algorithm solves an unconstrained but separable objective function,

$$\text{minimize } f(\mathbf{x}) + g(\mathbf{x}),$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex and differentiable, and $g : \mathbb{R}^n \mapsto \mathbb{R} \cup \{\infty\}$ is closed and convex but possibly nondifferentiable and has an inexpensive proximal mapping (Parikh et al., 2014). As introduced in Section 2.3.2, the proximal mapping of a closed proper convex function g is given by

$$\text{prox}_{\tau g}(\mathbf{v}) = \underset{\mathbf{x}}{\text{argmin}} \left(g(\mathbf{x}) + \frac{1}{2\tau} \|\mathbf{x} - \mathbf{v}\|_2^2 \right),$$

with parameter $\tau > 0$. The proximal gradient algorithm is then

$$\mathbf{x}^{(t+1)} = \text{prox}_{\tau g}(\mathbf{x}^{(t)} - \tau \nabla f(\mathbf{x}^{(t)})),$$

where t is the iteration counter (Parikh et al., 2014). To apply the proximal gradient algorithm to the dual problem (4.23), we identify

$$f(\boldsymbol{\lambda}) = \frac{1}{2} \|\mathbf{x} - \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m\|_2^2 \quad \text{and} \quad g_{\tilde{m}}(\boldsymbol{\lambda}_{\tilde{m}}) = \sum_{l \in \mathcal{E}_{\tilde{m}}} \delta_{C_{\tilde{m},l}}(\boldsymbol{\lambda}_{\tilde{m},l}),$$

for some $\tilde{m} = 1, 2, 3$. Then

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\lambda}_{\tilde{m}}} f(\boldsymbol{\lambda}) &= -\mathbf{A}_{\tilde{m}} \left[\mathbf{x} - \sum_{m \neq \tilde{m}} \mathbf{A}_m^T \boldsymbol{\lambda}_m \right] + \mathbf{A}_{\tilde{m}} \mathbf{A}_{\tilde{m}}^T \boldsymbol{\lambda}_{\tilde{m}} \\ &= \mathbf{A}_{\tilde{m}} \left[\sum \mathbf{A}_m^T \boldsymbol{\lambda}_m - \mathbf{x} \right], \end{aligned}$$

so the updates for the proximal gradient algorithm are

$$\boldsymbol{\lambda}_{\tilde{m}}^{(t+1)} = \text{prox}_{\tau g} \left(\boldsymbol{\lambda}^{(t)} - \tau \mathbf{A}_{\tilde{m}} \left[\sum \mathbf{A}_m^T \boldsymbol{\lambda}_m^{(t)} - \mathbf{x} \right] \right), \quad \tilde{m} = 1, 2, 3, \quad (4.24)$$

for a step size τ . In terms of implementation, it is worth noting that the updates (4.24) can be rewritten as

$$\boldsymbol{\lambda}_{\tilde{m}}^{(t+1)} = \text{prox}_{\tau g} \left(\boldsymbol{\lambda}^{(t)} + \tau \mathbf{A}_{\tilde{m}} \mathbf{u}^{(t)} \right), \quad \tilde{m} = 1, 2, 3,$$

where

$$\mathbf{u}^{(t)} = \mathbf{x} - \sum_m \mathbf{A}_m^T \boldsymbol{\lambda}_m^{(t)},$$

is the estimate of the primal variables at iteration t , which can be used to calculate the duality gap to monitor convergence. Convex triclustering has been implemented in MATLAB using the FASTA (Fast Adaptive Shrinkage/Thresholding Algorithm) toolbox (Goldstein et al., 2015), which is an implementation of the accelerated proximal gradient algorithm. Code for convex triclustering will be made available in a forthcoming MATLAB toolbox.

4.5 Practical Considerations

Before analyzing the performance of convex triclustering through simulations, we first address some important considerations for using the method in practice.

4.5.1 Specifying the Weights

The weights in the penalty terms can have a large effect on the quality of the clustering results, as observed by Chen et al. (2015) and consistent with our experience. In addition, the use of sparse weights can also lead to non-trivial improvements in the computational time (Chi & Lange, 2015; Chi et al., 2017). Our approach to constructing the weights follows closely to what has been recommended previously in the convex clustering literature (Chen et al., 2015; Chi et al., 2017). One noteworthy deviation is that, instead of using the original data tensor \mathbf{X} to construct the weights, we instead first construct a low-rank approximation of \mathbf{X} via the Tucker decomposition (4.9), denoted $\tilde{\mathbf{X}}$, and use the approximation to construct the weights. The use of a de-noised version of the data tensor improves the quality of the weights, which in turn can lead to a marked improvement in clustering performance (Section 4.6.4). One downside to incorporating the Tucker decomposition is that it introduces another tuning parameter, the rank of the decomposition. When applicable, a user can leverage problem-specific knowledge to select the rank for the decomposition. However, the availability of an automatic approach is desirable when such knowledge is unavailable. Several methods have been proposed for choosing the optimal rank of a Tucker decomposition, and based on our experiments we recommend using the SCORE algorithm by Yokota et al. (2017) to automatically select the rank for the decomposition (Section 4.6).

After a Tucker decomposition has been performed, we first calculate the “pre-weights”

$\hat{w}_{1,ij}^{(p)}$ between the i th and j th slices along mode 1 as proposed by Chen et al. (2015),

$$\hat{w}_{1,ij}^{(p)} = \iota_{\{i,j\}}^k \exp(-\phi \|\tilde{\mathbf{X}}_{i::} - \tilde{\mathbf{X}}_{j::}\|_{\mathbb{F}}^2). \quad (4.25)$$

The first term in equation (4.25), $\iota_{\{i,j\}}^k$, is an indicator function that equals 1 if the j th slice is among slice i 's k -nearest neighbors (or vice versa) and 0 otherwise. The purpose of this term is to control the sparsity of the weights. The corresponding tuning parameter k influences the connectivity of the underlying weights graph for mode m . One can explore different levels of granularity in the clustering by varying k (Chen et al., 2015). As a default, one can use the smallest k such that the underlying weights graph is still fully connected using either breadth-first search or depth-first search (Chen et al., 2015; Hopcroft & Tarjan, 1973). In MATLAB this can be accomplished using the `conncomp` function. Chi & Lange (2015) conjectured that one does not need to calculate the exact k -nearest neighbors, which scales quadratically in the number of fibers in the mode. A fast approximation to the k -nearest neighbors is likely sufficient for the sake of inducing sparsity into the weights (Indyk & Motwani, 1998; Muja & Lowe, 2009). The second term in equation (4.25) is the Gaussian kernel, which puts more fusion pressure on slices that are similar to one another. Intuitively, the weights should be inversely proportional to the distance between the i th and j th slices (Chen et al., 2015; Chi et al., 2017). The non-negative scale parameter in the Gaussian kernel, ϕ , controls the rate at which the pressure to fuse is applied to the i th and j th slices as a function of the distance between them. A value of $\phi = 0$ corresponds to uniform weights.

To obtain the final weights for mode 1, $\hat{w}_{1,ij}$, the pre-weights are normalized to sum to $1/\sqrt{I_2 I_3}$. The purpose of the normalization step is to keep the three penalty terms, $J_1(\mathbf{U})$, $J_2(\mathbf{U})$, and $J_3(\mathbf{U})$, on the same scale so that a single tuning parameter suffices

(Chi et al., 2017). Otherwise, one of the modes will dominate the clusterings and distort the results. Since the mode-1 slices are in $\mathbb{R}^{I_2 \times I_3}$, we choose the mode-1 weights to sum to $1/\sqrt{I_2 I_3}$, and the weights for the other modes are calculated analogously.

4.5.2 Choosing ρ

Convex triclustering estimates a sequence of clustering assignments as a function of one tuning parameter, ρ , which regulates the complexity of the solution, or number of triclusters. One obvious practical consideration is how to choose ρ to produce a final clustering result. In some applications, it may be suitable for a user to manually inspect the clustering sequence and use domain knowledge to pick ρ , especially given that clustering is an exploratory method. Since this approach is time consuming and requires expert knowledge, an automated, data-driven procedure for selecting ρ is desirable. In fact, the lack of a robust rule for choosing ρ has been cited as a drawback of convex clustering methods (Chrétien et al., 2016). Cross-validation (Stone, 1974; Geisser, 1975) and stability selection (Meinshausen & Bühlmann, 2010) are popular techniques for tuning parameter selection, but since both methods are based on resampling, they are unattractive in the tensor setting due to the computational burden. The use of information criteria, such as the Akaike information criterion (AIC) (Akaike, 1974) or the Bayesian information criterion (BIC) (Schwarz et al., 1978), is an attractive alternative as it does not rely on resampling and thus is not as computationally costly as cross-validation or stability selection. The BIC is given as

$$\text{BIC}(\rho) = n_T \log \left(\frac{\text{RSS}_\rho}{n_T} \right) + \text{df}_\rho \log(n_T), \quad (4.26)$$

where $n_T = I_1 I_2 I_3$ is the total number of elements in the data tensor, RSS_ρ is the residual sum of squares for a particular value of ρ and is calculated as $\|\mathbf{X} - \hat{\mathbf{U}}\|_F^2$, and df_ρ is the degrees of freedom for a particular value of ρ . We use the number of triclusters identified along the clustering path as an estimate of the degrees of freedom, df_ρ , which is consistent with the spirit of degrees of freedom since each tricluster mean is an estimated parameter. Looking at (4.26), we see that the BIC approach is in the same vein as the penalized estimation framework (1.7), as the BIC attempts to strike a balance between the accuracy of the fit (first term) and the complexity the model (second term).

The BIC is calculated on a grid of values for ρ , and the optimal ρ , denoted ρ^* , corresponds to the smallest value of the BIC. That is,

$$\rho^* = \min \text{BIC}(\rho). \quad (4.27)$$

One limitation to using the BIC is that it generally performs well only when the sample size is much larger than the number of parameters, so it is not suitable for our case (Chen & Chen, 2008, 2012). To overcome this drawback, Chen & Chen (2008, 2012) proposed the extended BIC,

$$\text{eBIC}_\alpha(\rho) = n_T \log \left(\frac{\text{RSS}_\rho}{n_T} \right) + \text{df}_\rho \log(n_T) + 2\alpha \text{df}_\rho \log(n_T), \quad (4.28)$$

where $\alpha \in [0, 1]$ is an additional tuning parameter. The eBIC (4.28) augments the classic BIC (4.26) with an additional term that further penalizes the complexity of the model in an attempt to overcome the BIC's poor performance for models with a relatively large number of parameters. The eBIC tuning parameter, α , governs how much weight the additional eBIC term carries. When $\alpha = 0$, we see that the extended BIC reduces to the

classic BIC. The performance of eBIC has been studied in a variety of settings, including linear regression (Chen & Chen, 2008), generalized linear models with (Chen & Chen, 2012) and without (Chen & Luo, 2013) canonical link functions, Gaussian graphical models (Foygel & Drton, 2010), ultra-high dimensional linear regression (Luo & Chen, 2013, 2014), convex clustering (Tan & Witten, 2015), and the Cox proportional hazards model (Luo et al., 2015). While these studies have demonstrated the eBIC’s improved performance over the classic BIC, the eBIC requires specifying the tuning parameter α . Based on the results in the literature and our initial experiments, we adopt a value of $\alpha = 0.5$.

4.6 Simulation Studies

To investigate the performance of convex triclustering in identifying triclusters in tensor data, we explore some simulated examples. For the majority of the simulations, we include two different variants of convex triclustering corresponding to slightly different approaches to constructing the weights as presented in Section 4.5.1. In both cases, a low-rank approximation of the noisy data tensor is constructed using the Tucker decomposition, where the only difference is the approach to selecting the rank of the decomposition. Choosing the optimal rank to use for either of the main tensor decompositions is a difficult and open question (Yokota et al., 2017; Kolda & Bader, 2009). While in some situations it makes sense for the rank to be user-specified based on problem-dependent knowledge, as with choosing the penalization parameter ρ an automatic procedure is desirable. During initial experiments, a few different methods for selecting the Tucker decomposition rank from the literature were compared: an L-curve approach that attempts to strike a balance between the decomposition’s relative error and compression ratio, as implemented by the `mllrankest` function in the `Tensorlab` MATLAB toolbox

(Vervliet et al., 2016), minimum description length (Rissanen, 1978; Yokota et al., 2017), and the recently-proposed SCORE algorithm (Yokota et al., 2017). Out of these, the use of the SCORE algorithm to select the Tucker decomposition rank for constructing the weights ultimately resulted in the best average triclustering performance. The SCORE algorithm itself includes a tuning parameter, $\hat{\gamma}$, and Yokota et al. (2017) suggest to use $\hat{\gamma} \in [0.0001, 0.01]$. We considered $\hat{\gamma} \in \{0.0001, 0.001, 0.01\}$ and found 0.001 to perform the best, which also matches the value used in the experiments by Yokota et al. (2017). In the simulation results, this variant of convex triclustering is referred to as `Cvxtriclustr TD1`.

During the initial experiments to compare different approaches for choosing the rank of the decomposition, a simple yet effective heuristic for choosing the rank was developed. For this heuristic, the Tucker rank for the m th mode is estimated by `floor($\sqrt{I_m}/2$)`, where the `floor` function rounds down to the nearest integer. Two principles motivating the heuristic are that the rank of the decomposition should be both small relative to and also in proportion to the length of the modes. Its inclusion in the simulations also serves as a robustness check to verify that convex triclustering’s performance does not crucially depend on the choice of the rank. We refer to this approach to constructing the weights as `Cvxtriclustr TD2` in the simulation results. Once the Tucker approximation to the data tensor has been constructed, it is then used to calculate the pre-weights using the Gaussian kernel (Section 4.5.1). The pre-weights are then filtered with k -nearest-neighbors and normalized as discussed in Section 4.5.1. Finally, ρ is chosen using the effective BIC from Section 4.5.2.

We compare convex triclustering to a k -means-based approach that has been used multiple times in the tensor clustering literature (Kutty et al., 2011; Liu et al., 2013; Zhang et al., 2013; Wu et al., 2016). This method, which we refer to as CPD+ k -means,

first performs a CP decomposition to reduce the dimensionality of the problem, and then applies k -means clustering to each factor matrix. k -means has also been used to cluster the factor matrices resulting from a Tucker decomposition (Acar et al., 2006; Sun et al., 2006; Kolda & Sun, 2008; Sun et al., 2009; Kutty et al., 2011; Liu et al., 2013; Zhang et al., 2013; Cao et al., 2015; Oh et al., 2017). We also considered this method in initial experiments, but its performance was inferior to that of CPD+ k -means so those results are omitted.

Both convex triclustering and CPD+ k -means have tuning parameters that need to be selected. For the rank of the CPD, we consider $R = \{2, 3, 4, 5\}$ and use the BIC formula from Sun et al. (2015) to automatically select the rank. A CP decomposition is then performed using the chosen rank, and those factor matrices are the input into the k -means algorithm to cluster along each mode. One of the drawbacks to using k -means is that k , the number of clusters, needs to be specified a priori. Several methods for selecting k have been proposed in the literature, and we use the gap statistic developed by Tibshirani et al. (2001) to select an optimal k^* from the specified possible values. Since convex triclustering estimates an entire solution path of triclustering results, ranging from $I_1 I_2 I_3$ triclusters to one big tricluster, we consider a rather large set of possible k values to make the methods more comparable.

To assess the quality of the clustering performance, we consider two measures that are common in the clustering literature. These are the adjusted Rand index (ARI) and variation of information (VI), both of which compare a method’s clustering result to the actual true clustering assignment. The ARI (Hubert & Arabie, 1985) varies between -1 and 1, where 1 indicates a perfect match, 0 corresponds to random clustering, and negative values indicate the clustering result is worse than what is expected from random partitioning. VI is based on information theory, and measures the amount of information

gained or lost in changing from one clustering to another (Meilă, 2007). VI attains its minimum value of 0 when the clustering is identical to the ground truth.

Since the original goal is to identify triclusters, most of the results presented in this section focus on the average triclustering performance across 200 simulated replicates. However, the mode-by-mode performance is also of interest since that is another area of the tensor clustering literature (see Section 4.3). As such, full simulation results are given in Appendix B. All simulations were performed in MATLAB using the `Tensor Toolbox` (Bader et al., 2015). Convex triclustering was implemented using the `FASTA` toolbox (Goldstein et al., 2015), which greatly reduced the algorithm development time.

4.6.1 Cubical Tensors, Checkbox Pattern

For the first and main simulation setting, we study clustering data in a cubical tensor generated by a basic checkbox mean model (Figure 4.6). The checkbox mean model is a generalization of the checkerboard mean model that has been studied in the context of biclustering (Madeira & Oliveira, 2004; Tan & Witten, 2014; Chi et al., 2017). Each entry in the observed data tensor is defined by the r th row group, the c th column group, and the t th tube group, plus Gaussian noise. That is, for the (i, j, k) entry of \mathcal{X} , we have

$$x_{ijk} = \mu_{rct} + \varepsilon_{ijk}, \quad (4.29)$$

where the grand mean is assumed to be zero for identifiability and $\varepsilon_{ijk} \sim N(0, \sigma_{rct}^2)$ for some $\sigma_{rct}^2 > 0$. Unless specified otherwise, there are two true clusters along each mode for a total of eight triclusters. For now, we generate only cubical tensors to keep things simple, and the impact of the tensor’s shape on clustering performance will be studied in Section 4.6.2. For some settings, we restrict our attention to low and high noise

situations. A low noise, or conversely high signal-to-noise ratio (SNR) setting, is such that the signal of 1 is comparable to the noise ($\sigma = 3$), while a low SNR corresponds to $\sigma = 6$. These SNR values were chosen to be consistent with the ones used by Chi et al. (2017) to study convex biclustering.

Balance and Homoskedasticity

To get an initial feel for how the different triclustering methods perform at recovering the true underlying checkbox structure, we first consider a situation where the clusters corresponding to the two classes along each mode are all equally-sized, or balanced, and share the same error variance. The average triclustering performance for this setting in a tensor with dimensions $I_1 = I_2 = I_3 = 60$ are given in Figure 4.8 for different noise levels. The story across ARI and VI is largely similar, so we will focus on ARI. As can be seen in Figure 4.8a, all three methods perform well when the noise level is low ($\sigma = 1$). As the noise level increases, however, CPD+ k -means experiences an immediate and noticeable dropoff in performance. Convex triclustering, on the other hand, is able to maintain near-perfect performance until the noise level becomes rather high ($\sigma = 8$).

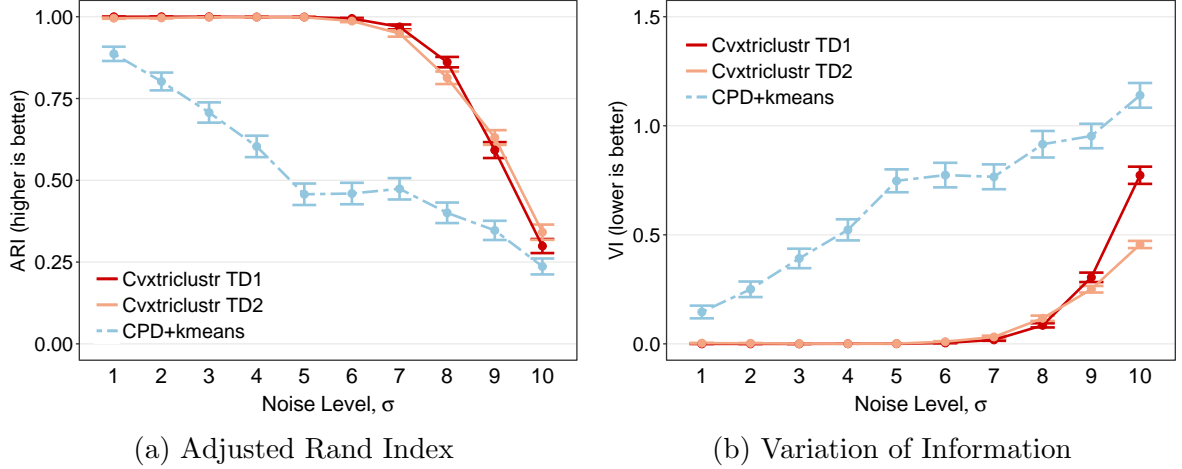


Figure 4.8: Checkerbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error.

Appendix B.1 contains graphs of the average clustering performance for different tensor sizes in this setting. As to be expected, the performance for different levels of noise depends on the size of the tensor. The noise level at which convex triclustering is no longer able to maintain perfect triclustering is smaller ($\sigma = 5$) with a smaller tensor of size $I_1 = I_2 = I_3 = 40$, but intuitively the method can withstand more noise in a larger tensor (Figures B.1d and B.5d). From these graphs we see that there also is some variability in performance across modes for both convex triclustering and CPD+k-means, despite dealing with a cubical tensor and balanced cluster sizes along each mode. For example, convex triclustering performs the worst along mode 2 for $I_1 = I_2 = I_3 = 40$, while this is actually the strongest mode for CPD+k-means.

Imbalanced Cluster Sizes

When comparing clustering methods, one factor of interest is the extent to which the relative sizes of the clusters impact clustering performance. To investigate this, we again

use a cubical tensor of size $I_1 = I_2 = I_3 = 60$ but introduce different levels of cluster size imbalance along each mode, as measured by cluster 2's size (n_{m2}) relative to the length of the mode, n_{m2}/I_m for $m = 1, 2, 3$. When the noise is relatively low, CPD+ k -means is largely unaffected by the imbalance until the size of cluster 2 is less than 30% of the mode's length (Figures 4.9a and 4.9b). At this point, its triclustering performance drops off significantly and it performs roughly the same as a random clustering assignment when the sizes are highly skewed ($n_{m2}/I_m = 0.1$). Convex triclustering is more or less invariant to the imbalance as its performance is almost perfect across all levels of cluster size imbalance. It experiences a slight deterioration in performance only for $n_{m2}/I_m = 0.1$ in the high noise case.

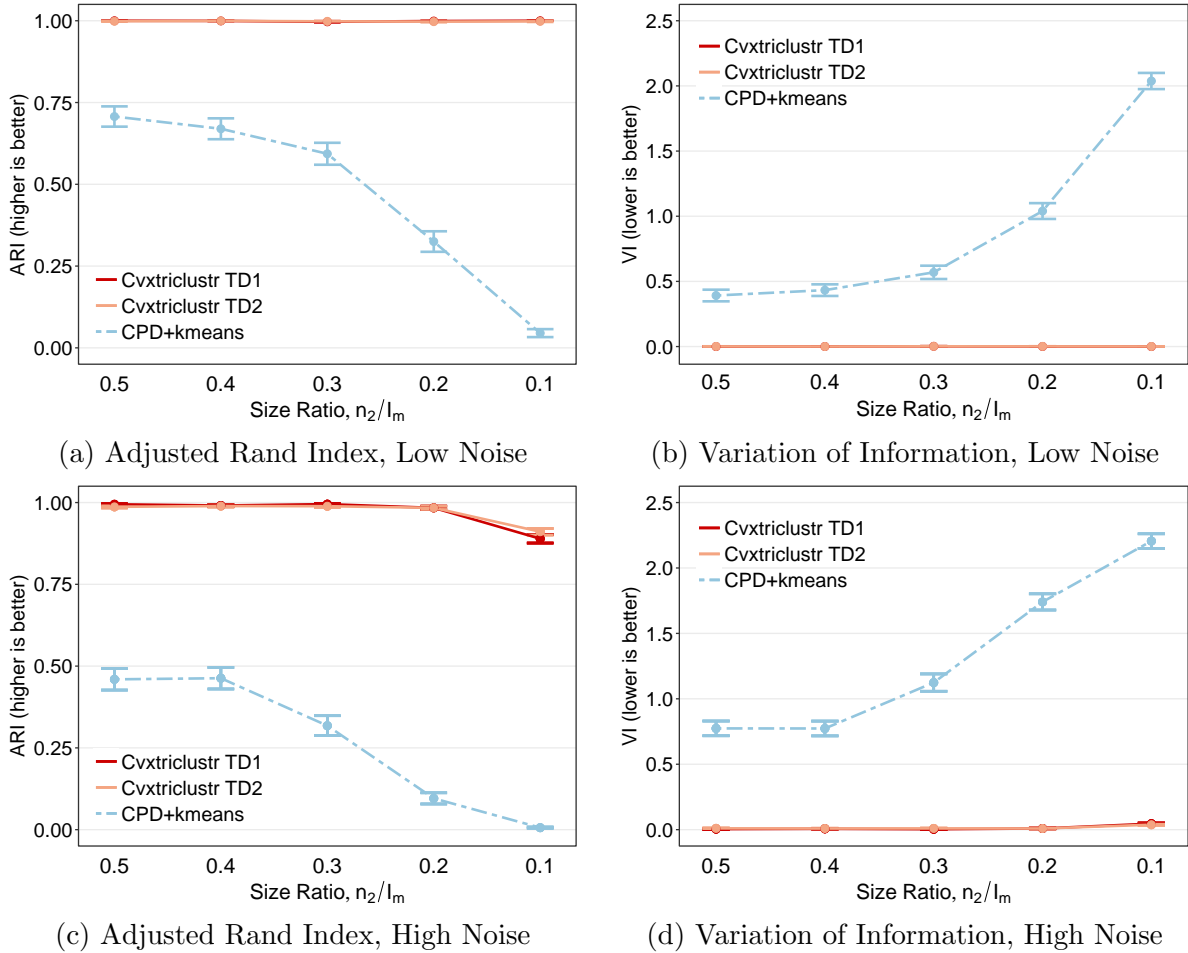


Figure 4.9: Checkerbox Simulation Results: Impact of Cluster Size Imbalance. Two imbalanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different degrees of cluster size imbalance. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$). Size ratio = 0.5 corresponds to balanced clusters.

Heteroskedasticity

Another factor of interest is how the clustering methods perform when there is heteroskedasticity in the variability of the two classes. Figure 4.10 displays the triclustering performance for different degrees of heteroskedasticity, as measured by the standard de-

viation for class 2 relative to class 1's standard deviation, σ_2/σ_1 . In the low noise setting, convex triclustering is immune to the heteroskedasticity until the noise levels differ by a factor of 4. CPD+ k -means instead is very sensitive to a departure from homoskedasticity, experiencing a decline even when the noise ratio increases slightly from 1 to only 1.5. Convex triclustering fares worse in the high noise setting and also has a drop in performance with a small deviation from homoskedasticity. Once class 2's standard deviation is more than double the standard deviation for class 1, all three methods are essentially the same as random clustering. This result is not terribly surprising since, in the high noise setting, this would result in one class having a very high standard deviation of $\sigma_2 > 12$.

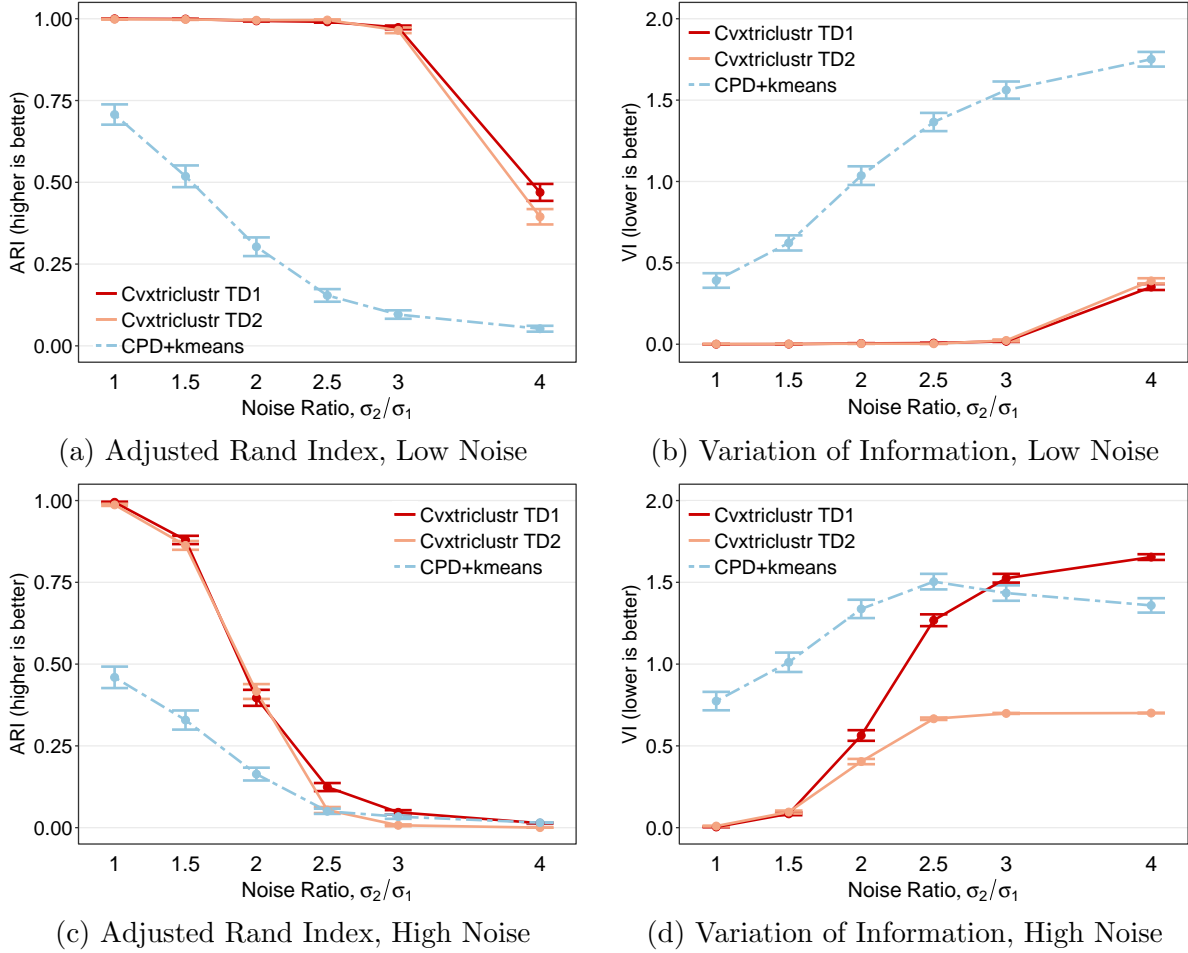


Figure 4.10: Checkerbox Simulation Results: Impact of Heteroskedasticity. Two balanced clusters per mode with either low or high heteroskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different levels of heteroskedasticity. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$). Noise ratio = 1 corresponds to homoskedastic noise.

Different Clustering Structures

So far, we have considered a simple situation where there are exactly two true clusters along each mode, for a total of eight triclusters. Another factor of practical importance is how the clustering methods perform when there are more than two clusters per mode,

and also when the number of clusters along each mode differs. We investigate both of these settings in this section. As before, the tensor is a perfect cube with $I_1 = I_2 = I_3 = 60$ observations along each mode and an underlying checkbox pattern. To gauge performance, we again focus the attention on how the methods perform in the presence of both low ($\text{SNR} = 1/3$) and high ($\text{SNR} = 1/6$) noise.

The first situation studied is one in which there are three true clusters along each mode, resulting in a total of 27 triclusters. The results from this simulation setting are given in the left hand side of each graph in Figure 4.11. From the graphs it can be seen that convex triclustering severely outperforms CPD+ k -means in this setting, and the dominance is consistent across both noise levels. Convex triclustering is able to recover the true triclusters almost perfectly, while CPD+ k -means struggles mightily to handle the increased number of clusters per mode.

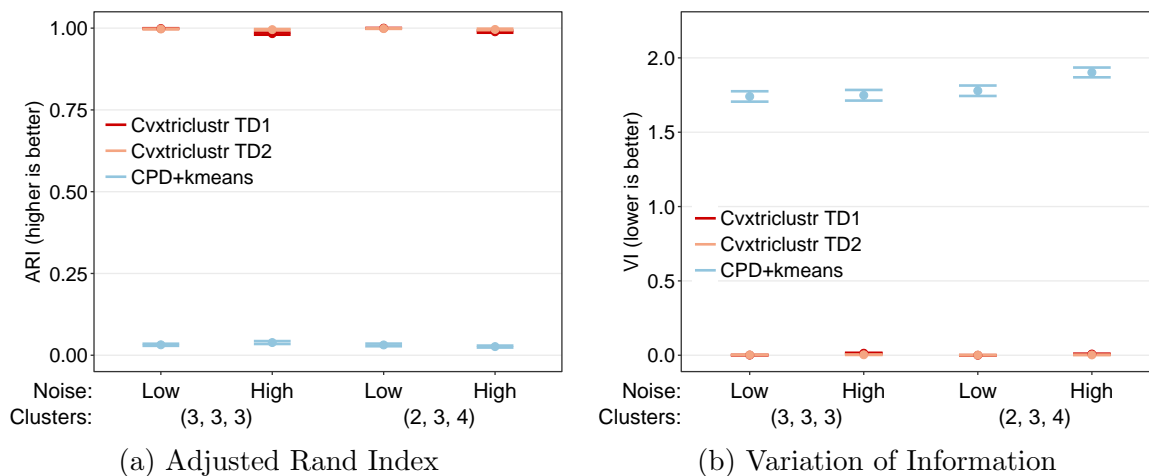


Figure 4.11: Checkerbox Simulation Results: Impact of Clustering Structure. Different number of balanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different clustering structures, corresponding to either three clusters per mode or two, three, and four clusters along modes one, two, and three. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$).

We also investigated the clustering performance when the number of clusters per mode varies. In this setting, there are two, three, and four clusters along modes one, two, and three, respectively. From the right hand side of each graph in Figure 4.11, we can see that the results are similar to the situation with three clusters per mode. CPD+ k -means again performs very poorly across both noise levels, while convex triclustering is again able to essentially recover the true triclustering structure. Compared to the setting with three clusters per mode, CPD+ k -means performs slightly worse in the face of a more complex clustering structure, while convex triclustering is able to handle it in stride. These results are very favorable for convex triclustering as the basic clustering structure of only two clusters per mode is unlikely to be observed in practice.

4.6.2 Rectangular Tensors

Up to this point, to get an initial feel for convex triclustering’s performance, we restricted our attention to cubical tensors with the same number of observations per mode so as to avoid changing too many factors at once. However, this was a simplifying assumption and it is unlikely that the data tensor at hand will be a perfect cube, so it is important to understand the clustering performance when the methods are applied to rectangular tensors. The graphs in this section contain the results only for the adjusted rand index (ARI) since the story for variation of information is again similar, but the VI graphs can be found Appendix B.5. The first rectangular tensor is one in which there are two short modes ($I_1 = I_2 = 10$) and one relatively longer mode ($I_3 = 50$), and the clustering results for this tensor shape are presented Figure 4.12.

At a lower noise level ($\sigma = 2$), convex triclustering performs very well and outperforms CPD+ k -means in terms of both single-mode clustering and triclustering. When the noise level is bumped up ($\sigma = 3$), both methods experience a noticeable drop off in their

performance and now perform more similarly. Interestingly, convex triclustering’s single-mode clustering results are better along the two shorter modes (modes 1 and 2), which is not what we expected. This provides some evidence that the performance along a mode depends on both the length of that mode as well as the lengths of the other modes. Another thing to note from this setting is that, along the two shorter modes, the use of the heuristic in determining the rank of the Tucker decomposition for calculating the weights performs better than the SCORE algorithm, though ultimately the triclustering performance is comparable. This may indicate that the SCORE algorithm struggles to correctly identify the optimal Tucker rank for short modes in the presence of relatively higher noise, while the heuristic is more immune to the noise level as it is based simply on the dimensions of the tensor. When the length of the shorter modes is increased slightly (from $I_m = 10$ to $I_m = 20$ for $m = 1, 2$), convex triclustering has near-perfect performance while CPD+ k -means performs roughly the same as before. Thus, convex triclustering struggles with this tensor shape only when the short modes are really short (only 10 observations).

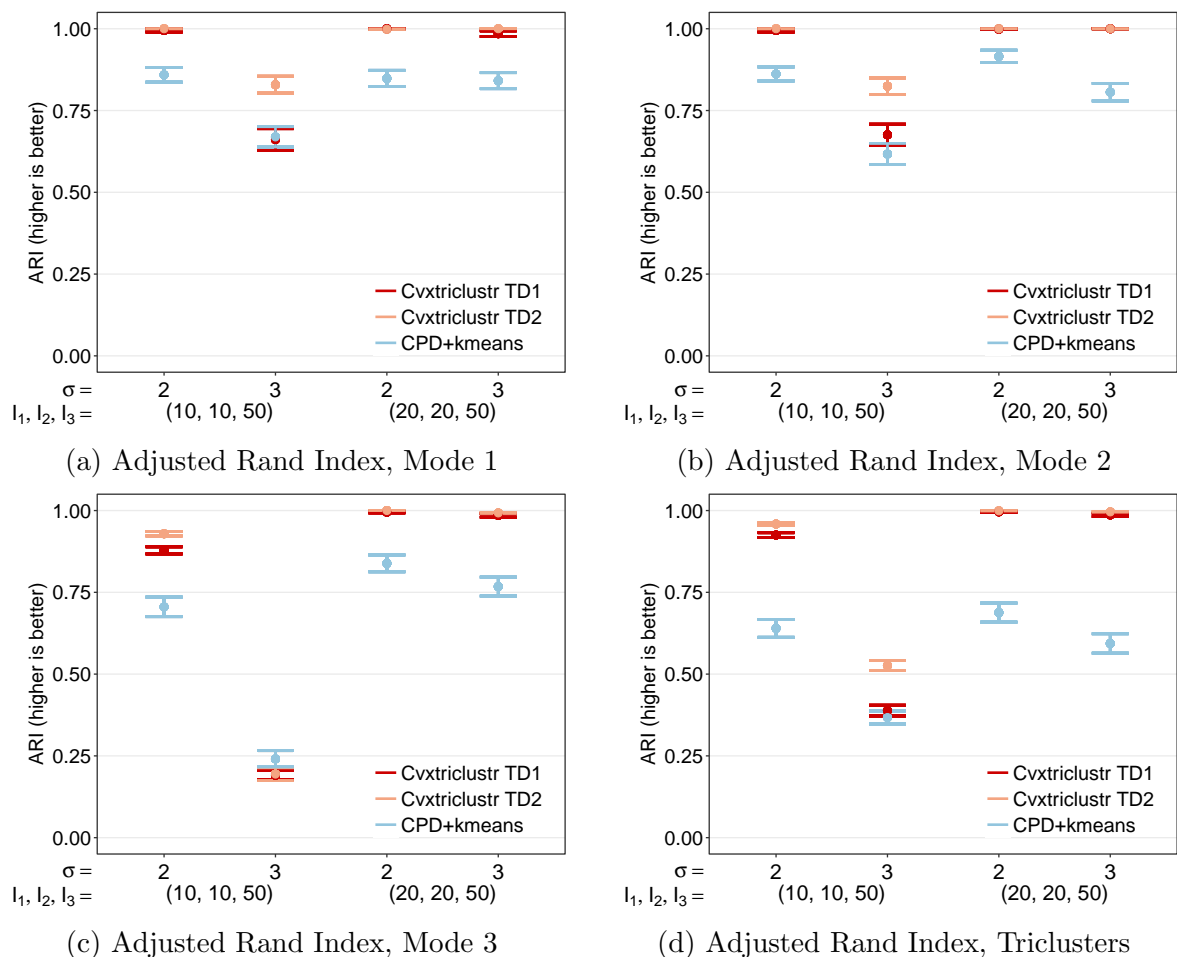


Figure 4.12: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with two short modes and one longer mode. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.

Now we turn to clustering a rectangular tensor with one short mode and two longer modes, with the results displayed in Figure 4.13. This setting was motivated by the results from the previous tensor shape (short-short-long) since the mode-by-mode performance was not as expected. As with that tensor shape, convex triclustering performs very well and better than CPD+k-means at the lower noise level ($\sigma = 3$), but again has a

sharp decrease in ARI at the higher noise level ($\sigma = 4$). Also like before, the decline is more pronounced for the longer modes (modes 2 and 3) as the short mode is still able to maintain perfect performance despite the increase in noise. The deterioration is much less pronounced for the slightly larger tensor ($I_1 = 20, I_2 = I_3 = 50$) and convex triclustering has much better triclustering performance than CPD+ k -means.

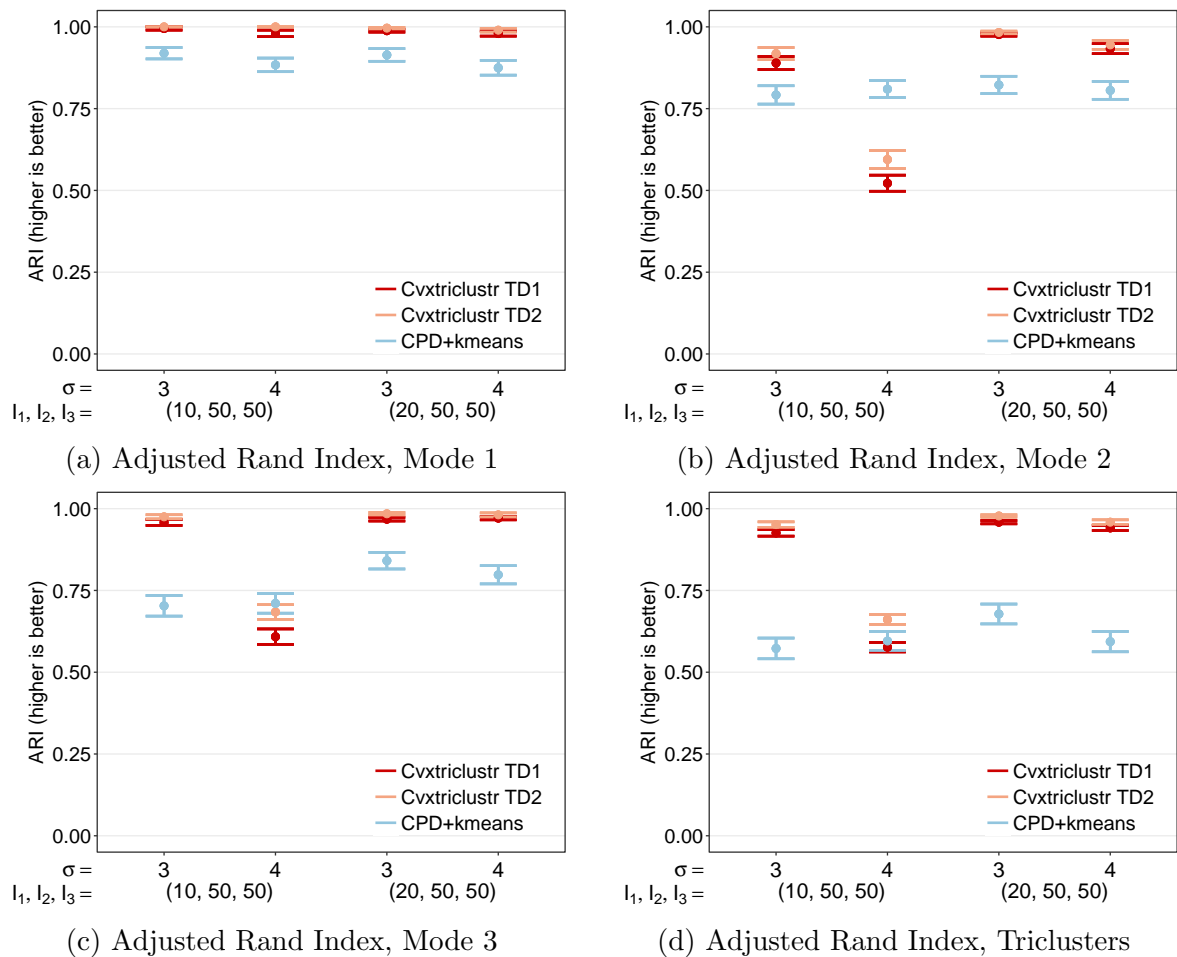


Figure 4.13: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with one short modes and two longer modes. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.

To further investigate the mode-by-mode performance with rectangular tensors, we also apply the clustering methods to a “Goldilocks” tensor with mode lengths that are short, medium, and long. This setting was again motivated by the results from the previous two tensor shapes to see how the performance is impacted when the size of a longer mode is increased. The ARI results for this tensor shape are given in Figure 4.14, and they are consistent with what was observed previously. When the short mode has only 10 observations, convex triclustering initially performs very well until the noise reaches a certain level. At this point, its performance for the longer modes declines sharply and actually performs worse than CPD+ k -means, and this pattern is more pronounced for the longest mode ($I_3 = 100$). The overall triclustering performance for both methods remains similar, however. As before, convex triclustering does not experience as much of a decrease when the shortest mode is made slightly longer ($I_1 = 20$), and for the most part still does noticeably better than CPD+ k -means.

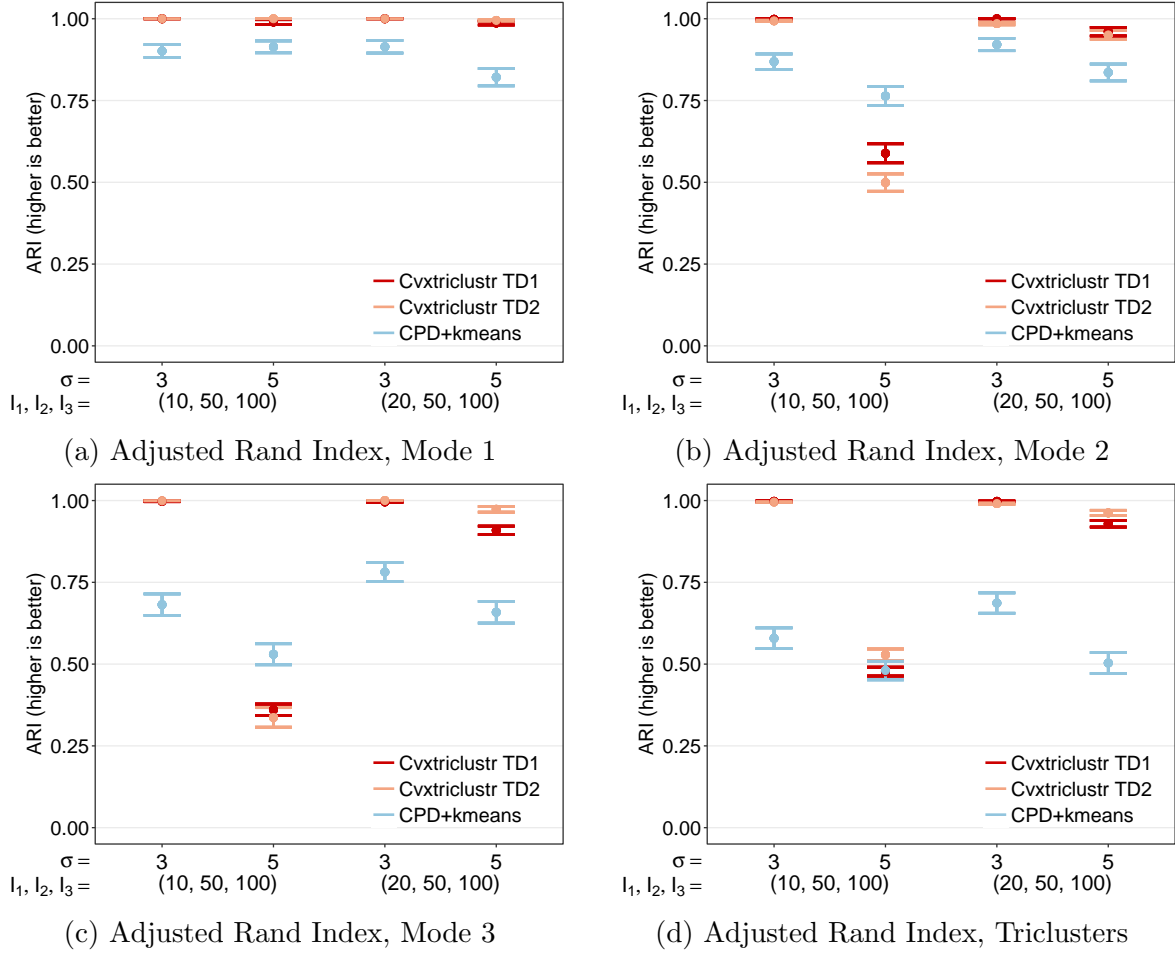


Figure 4.14: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with short, medium, and long mode lengths. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.

Overall, from clustering these different tensor shapes we see that convex triclustering still generally performs very well and better than CPD+ k -means. The main issue it encounters is when at least one mode is very short ($I_m \approx 10$). Convex triclustering performs very well at lower noise levels but has a sharp decline in performance once the noise reaches a certain level. Unexpectedly, the decline in single-mode performance

is worse for the longer modes. However, even when this happens, convex triclustering’s overall triclustering performance is still comparable to CPD+ k -means. Additionally, this pattern is much less striking when the length of the shortest mode is increased slightly.

4.6.3 CANDECOMP/PARAFAC Model

In Section 4.6.1 it was demonstrated that convex triclustering performs well and typically better than CPD+ k -means when clustering third-order tensors whose triclusters have an underlying checkbox pattern. The generative model for our second simulation setting is the CANDECOMP/PARAFAC (CP) model, which is based on the CP decomposition (Section 4.2.3) and is a commonly used model in the analysis of tensor data (Kolda & Bader, 2009; Acar & Yener, 2009). For a given rank $R \in \mathbb{Z}^+$, the CP model represents the data tensor as the sum of R rank-one tensors. For a third-order tensor, the rank-one tensors are the outer products of the columns in the underlying factor matrices,

$$\mathbf{X} \approx \sum_{r=1}^R \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r. \quad (4.30)$$

Thus for this simulation setting, we first construct the factor matrices, use the factor matrices to generate the true tensor as in equation (4.30), and then add varying levels of Gaussian noise to the true tensor to generate the observed data tensor. We consider two different types of factor matrices, both of which are based on classic non-convex cluster shapes that have been studied in the convex clustering literature before. As shown in Figure 4.15, one shape consists of two half-moon clusters (Hocking et al., 2011; Chi & Lange, 2015; Tan & Witten, 2015) while the other shape contains a bullseye, similar to the two-circles shape studied by Ng et al. (2002) and Tan & Witten (2015).

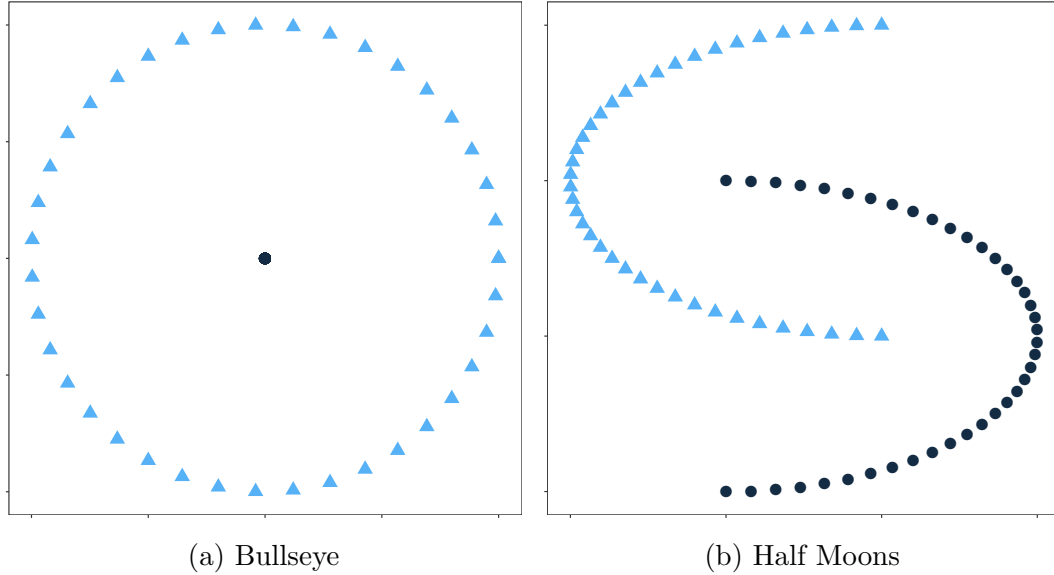


Figure 4.15: Factor Matrices for the CP Models.

The simulation results from using the CP model with non-convex shapes to generate the data are given in Figure 4.16. One thing that jumps out from the graphs is the large discrepancy in performance between CPD+ k -means and convex triclustering. Convex triclustering is able to almost perfectly identify the true triclusters but CPD+ k -means performs very poorly, even with the low noise level. The poor performance of CPD+ k -means is not a complete surprise as other researchers have noted the difficulty that k -means methods have in recovering non-convex clusters (Ng et al., 2002; Hocking et al., 2011; Tan & Witten, 2015). However, it is encouraging that convex triclustering is able to still perform very well for these settings since the true triclusters do not have a checkbox pattern.

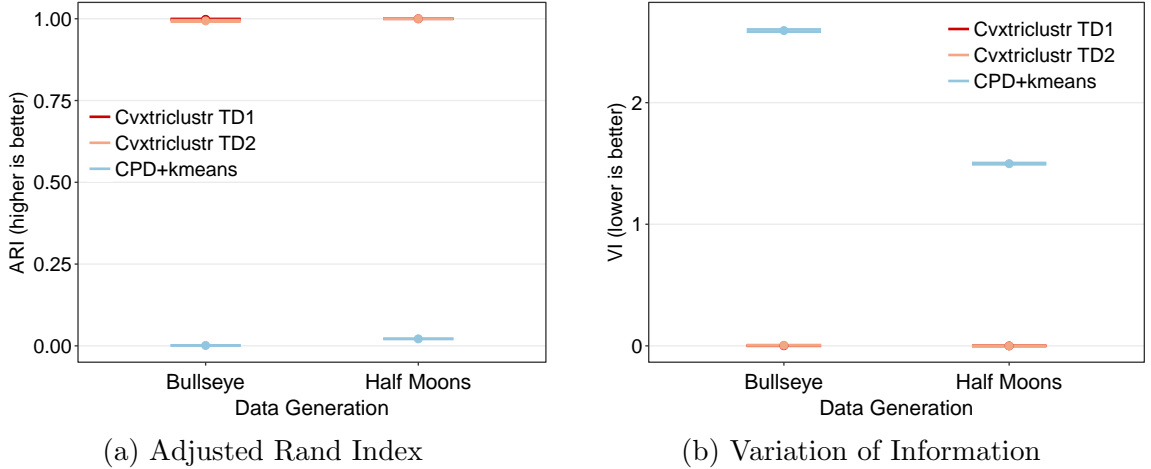


Figure 4.16: CP Model Simulation Results. Two balanced clusters per mode with low homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average triclustering performance plus/minus one standard error for two different data generation approaches. “Bullseye” and “Half Moons” refer to the shape embedded in the factor matrices used to generate the true tensor (Figure 4.15).

4.6.4 Importance of Good Weights

In this section we investigate the importance of constructing useful weights (Section 4.5.1) when using convex triclustering. To do so, we return to the original simulation setting of clustering a cubical tensor of size $I_1 = I_2 = I_3 = 60$ generated by the checkbox mean model (4.29) with two balanced clusters per mode and homoskedastic noise (Section 4.6.1). In addition to using the Tucker decomposition to construct the weights (Cvxtriclustr TD1), we also include weights constructed by the data tensor \mathcal{X} (Cvxtriclustr Data) and the true mean tensor \mathbf{U} (Cvxtriclustr True). CPD+kmeans is again included as a comparison, and the performance metrics for these methods are given in Figure 4.17.

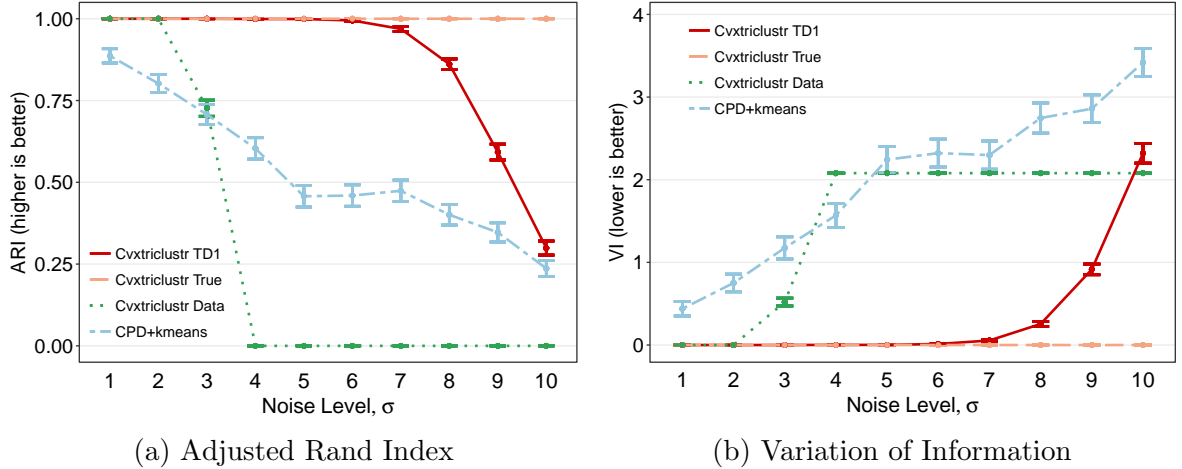


Figure 4.17: Impact of Convex Triclustering Weights. Comparison of different weights for convex triclustering for clustering a cubical tensor with two balanced clusters per mode and homoskedastic noise with $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different noise levels. TD1 refers to using a Tucker decomposition with the rank chosen using the SCORE algorithm (Section 4.5.1. True uses the true mean tensor \mathbf{U} to construct the weights, while Data uses the noisy data tensor \mathcal{X} .

Initially, when the noise is low ($\sigma \leq 2$), the three different weights with convex triclustering all perform perfectly. As the noise increases, the performance of convex triclustering with the data-based weights (Cvxtriclustr Data) quickly plummets and is equivalent to random clustering for $\sigma \geq 4$. By using the Tucker decomposition to construct a de-noised estimate of the data tensor in order to calculate the weights (Cvxtriclustr TD1), convex triclustering is much more robust to the increase in the noise and performs almost perfectly until $\sigma > 7$. Using weights based on the completely noiseless true tensor (Cvxtriclustr True) makes convex triclustering even more robust to the noise, as it is able to maintain perfect clustering even at very high noise levels. Although these weights are not a realistic option in practice, these results highlight the importance of constructing quality weights for convex triclustering’s performance, which was also observed by Chen et al. (2015) and Chi & Lange (2015). These results also reiterate the current gap

in the convex clustering literature between the theory, which uses non-sparse unit weights (Tan & Witten, 2015), and the empirical performance, which requires sparse and useful weights. Filling in this gap is an area of future research.

4.7 Real Data Application

Now that we have extensively studied the performance of convex triclustering in a variety of simulated settings, we turn to using our method on a real dataset. The proprietary dataset comes from a major online company and contains the click-through rates for advertisements displayed on the company’s webpages from May 19, 2016 through June 15, 2016. A click-through rate is calculated by dividing the number of times a user clicks on a specific advertisement by the number of times the advertisement was displayed. The dataset contains information on 1000 users, 189 advertisements, 19 publishers, and 2 different devices, aggregated across time. Thus the data form a fourth-order tensor where each entry in the tensor corresponds to the click-through rate for the given combination of user, advertisement, publisher, and device. Here a publisher simply refers to a different webpage within the online company’s website, such as the main home page versus a page devoted to either breaking news or sports scores. The two device types correspond to how the user accessed the page, using either a personal computer or a mobile device such as a cell phone or tablet computer.

Besides the general challenge of working with data arranged as a tensor, another challenge posed by the dataset is a plethora of missing values. If a specific advertisement is never seen by a user, it is considered a missing value since it is different from a value of zero that corresponds to a displayed advertisement that was not clicked on. Click-through rate data contain a lot of missing values since a given user likely has seen only a handful of the many different possible advertisements. For example, over 99% of the values in

the dataset at hand are missing. Since convex triclustering can only handle complete data, we first need to impute the missing values before any clustering can be done. To do so, we use the tensor completion method proposed by Jain & Oh (2014) that is based on the CP decomposition (Section 4.2.3). As is typical with tensor decompositions, their method requires that the rank of the decomposition be specified beforehand. To determine the rank to use, we perform tensor completion on the fourth-order data tensor using $\text{rank} = R \in \{1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 16, 18, 20, 22\}$ and select the optimal rank as the one that minimizes the BIC for tensor decompositions that was proposed by Sun et al. (2015). This procedure resulted in choosing a final rank of $R = 20$ for completing the missing values. Since click-through data are proportions, they live in the unit interval $[0, 1]$, but there is no guarantee in the tensor completion algorithm that the filled-in values will also be in $[0, 1]$. Therefore, as a post-processing step, we threshold the values to be in the unit interval as was done when this dataset was analyzed by Sun et al. (2015). One mode of the fourth-order tensor has only two observations and those observations already have a natural grouping (device type). Therefore, for the sake of clustering we analyze the devices separately. As with the simulations, we compare our method with CPD+ k -means. Furthermore, the number of clusters for convex triclustering is automatically selected using the extended BIC (Section 4.5.2) while the gap statistic (Tibshirani et al., 2001) does so for CPD+ k -means.

We first look at the clustering results from clustering the click-through rates for users accessing the advertisements through a personal computer (PC). Table 4.1 contains the number of clusters identified as well as the sizes of the clusters, while Figure 4.18a visualizes the advertisement-by-publisher biclusters for a random user. As to be expected, the advertisement-by-publisher slices display a checkerboard pattern, which turns into a checkbox pattern when the slices are meshed together. The clustering results for

the users are omitted as clustering the advertisements and the publishers was the main interest of the analysis. However, clustering the tensor does not result in the loss of information that would occur if the tensor was converted into a matrix by averaging across users or flattening along one of the modes. In Table 4.1 and Figure 4.18a, we see that convex triclustering identifies four advertisement clusters, with one cluster being much bigger than the others. The advertisements in this large cluster have click-through rates that are close to the grand average in the dataset. One of the small clusters has very low click-through rates, while the other two clusters tend to have much higher click-through rates than the rest of the advertisements. On the other hand, CPD+ k -means clusters the advertisements into 57 groups, which is less-useful from a practical standpoint. Many of the clusters are similarly-sized and contain only a few advertisements, likely due to the inability of CPD+ k -means to handle imbalanced cluster sizes as was seen in the simulation experiments (Section 4.6.1). In terms of the publishers, convex triclustering identifies three clusters while CPD+ k -means does not find any underlying grouping and simply identifies one big cluster, which again is not very useful (Table 4.1). One way online advertisers can reach more users is by entering agreements with other companies to route traffic to the advertiser’s website. For example, Google and Apple have a revenue-sharing agreement in which Google pays Apple a percentage of the revenue generated by searches on iPhones (McGarry, 2016). Similarly, the online company being studied partners with several internet service providers (ISPs) to host the default home pages for the ISP’s customers. It would make sense that these slightly different variants of the online company’s main home page would have similar click-through rates, and they were in fact all grouped into the same cluster by convex triclustering.

For users accessing the advertisements through a mobile device, such as a mobile phone or tablet computer, the convex triclustering results for the advertisements are

Device	Convex Triclustering				CPD+kmeans	
	Advertisements		Publisher		Advertisements	Publisher
	# of clusters	Cluster Sizes	# of clusters	Cluster Sizes	# of clusters	# of clusters
PC	4	(156, 22, 8, 3)	3	(4, 3, 12)	57	1
Mobile	3	(145, 22, 22)	2	(7, 12)	49	13

Table 4.1: Advertising Data Clustering Results

largely similar to the results for PCs (Table 4.1 and Figure 4.18b). There is one large cluster that contains click-through rates similar to the overall average, while the two other equally-sized clusters have relatively very low or very high click-through rates, respectively. The underlying click-through rates for the PC data have more variability than the mobile data, which is consistent with the identification of an additional cluster for the PC data. As before, CPD+ k -means finds a large number of advertisement clusters, most of which are roughly the same size, again likely impacted by the imbalance in the cluster sizes. When compared to the personal computer device, one difference is that the cluster with the higher click-through rates for mobile devices is larger and has a higher average click-through rate than the similar clusters for the personal computer device. This finding is consistent with research by the Pew Research Center that found that click-through rates for mobile devices are higher than for advertisements viewed on a personal computer or laptop (Mitchell et al., 2012).

It is also enlightening to take a closer look at the underlying advertisements clustered across the two devices. All of the advertisements clustered in the high click-through rate cluster for the mobile devices are in the average click-through rate cluster for personal computers. In taking a closer look at the ads in these clusters, there are several ads related to online shopping for personal goods, such as jeans, workout clothes, or neckties. It makes sense to shop for these types of goods using a mobile device, such as while

at work when it is not appropriate to do so on a work computer. Conversely, all of the advertisements in either of the two higher PC click-through rate clusters are in the large, average click-through rate cluster for the mobile devices. There are several financial-related ads in these two PC clusters, such as for mortgages or general investment advice. There are not many online shopping ads in those clusters, with the exception of more expensive technology-related goods that one may want to invest more time in researching before making a purchase.

In terms of the publishers, convex triclustering identifies two clusters while CPD+ k -means identifies 13 small clusters (Table 4.1). Contrary to the advertisement clusters, the publisher clusters across both devices are very similar. In fact, the only difference is that the smaller cluster for the mobile device, which contains seven publishers, is split into two clusters for personal computers. This can be seen in the click-through rate heatmaps given in Figure 4.18 in looking at the right part of each heatmap. The publishers in these smaller clusters have higher click-through rates on average than those in the larger cluster. Additionally, five of the seven (71%) publishers in the high click-through rate clusters have stand-alone apps that display ads, while only three of the twelve (25%) publishers in the larger cluster do. For mobile devices, it has been observed that in-app advertisements have higher click-through rates than browser-based ads (Hof, 2014). We conjecture that this is also true for personal computer apps, which is consistent with the clustering results. Thus it again appears that the clusters identified by convex triclustering also make sense practically.

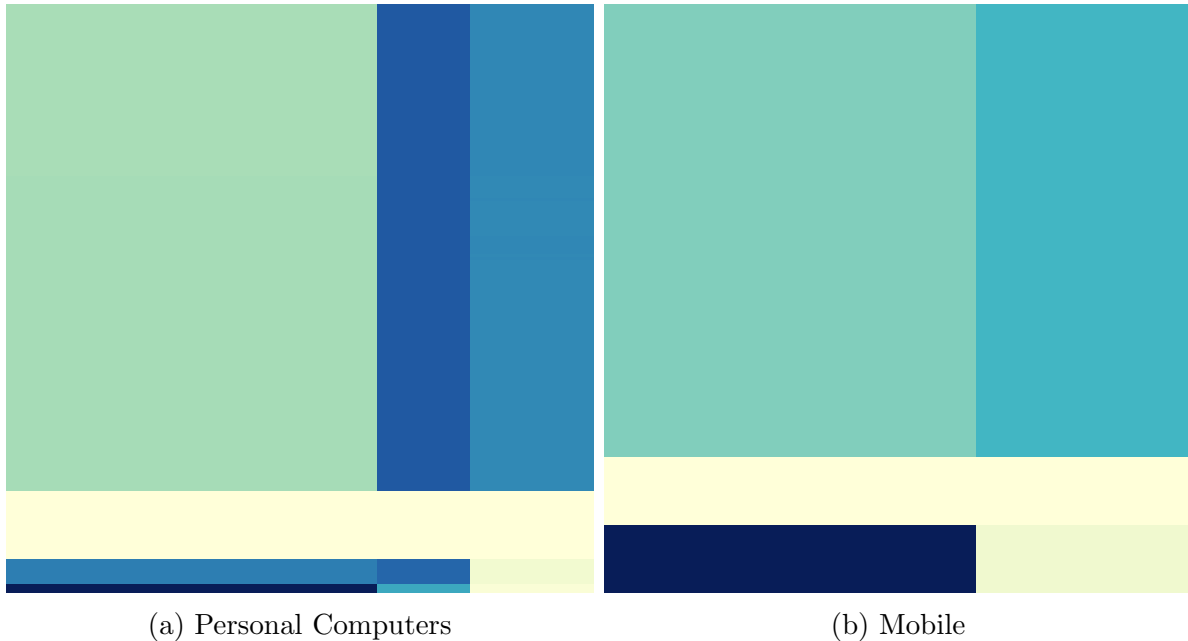


Figure 4.18: Advertisement and Publisher Click-Through Rate Biclusters for a Random User. Advertisements are on the y -axis and publishers are on the x -axis. Darker blue corresponds to higher click-through rates for a given device.

4.8 Discussion

In this chapter, we proposed and studied triclustering as a convex optimization problem. The convex formulation affords the method nice properties that remedy shortcomings of some of the existing methods. Convex triclustering always returns the unique global minimum regardless of the initialization, and its solutions are stable with respect to small changes in the data. Another hallmark of convex triclustering is its simplicity, in that the resulting solution path of triclusters is governed by only one tuning parameter and a principled method for automatically choosing the tuning parameter is in place. Additionally, an automated procedure for constructing the weights to fine-tune the clustering was also developed. Through a battery of simulated settings we see that convex

triclustering performs very well and typically better than the commonly-used CPD+ k -means approach. The simulation results also demonstrate the versatility of our method, in that it can be used for either single-mode clustering or triclustering. Convex triclustering also produced sensible results when applied to an online advertising dataset. Code for implementing convex triclustering in MATLAB will be available in a forthcoming toolbox.

There are several possible extensions that have been left for future work. The penalty terms in our formulation do not assume any natural ordering to the slices along a given mode. However, such a natural ordering can sometimes exist, such as when one of the modes represents time. For a situation like this, a fused-lasso type penalty that respects the ordering by fusing together only adjacent slices may improve the clustering results. Another way in which the clustering results could potentially improve is through the use of different weights. As noted, the performance of convex triclustering is somewhat sensitive to the choice of the weights. Although we have developed an automatic method for constructing weights that work well empirically, exploring other approaches to constructing the weights is a direction of future research. For example, other tensor denoising methods, such as the use of the ℓ_1 norm to make the decomposition most robust to noise as done by Cao et al. (2015), could possibly improve the quality of the weights.

Algorithmically, a distributed implementation of convex triclustering would increase its practical usability. A natural approach would be to adopt an existing distributed version of the proximal gradient method, such as one of the methods proposed by Combettes & Pesquet (2011), Chen & Ozdaglar (2012), or Li et al. (2013). Such a development would likely aid in the ability of the convex triclustering methodology to be extended to higher-order tensors as well, which is also of interest.

REFERENCES

- Acar, E., Çamtepe, S. A., & Yener, B. (2006). Collective sampling and analysis of high order tensors for chatroom communications. In *International Conference on Intelligence and Security Informatics*, (pp. 213–224). Springer.
- Acar, E., & Yener, B. (2009). Unsupervised multiway data analysis: A literature survey. *IEEE transactions on knowledge and data engineering*, *21*(1), 6–20.
- Ahmed, H. A., Mahanta, P., Bhattacharyya, D. K., & Kalita, J. K. (2014). Shifting-and-scaling correlation based biclustering algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, *11*(6), 1239–1252.
- Akaike, H. (1974). A new look at the statistical model identification. *IEEE transactions on automatic control*, *19*(6), 716–723.
- Allen, G. I., & Liu, Z. (2013). A local poisson graphical model for inferring networks from sequencing data. *IEEE transactions on nanobioscience*, *12*(3), 189–198.
- Altenbuchinger, M., Rehberg, T., Zacharias, H., Stämmler, F., Dettmer, K., Weber, D., Hiergeist, A., Gessner, A., Holler, E., Oefner, P. J., et al. (2016). Reference point insensitive molecular data analysis. *Bioinformatics*, *33*(2), 219–226.
- Amiri, S., Clarke, B. S., & Clarke, J. L. (in press). Clustering categorical data via ensembling dissimilarity matrices. *Journal of Computational and Graphical Statistics*.
- Arnold, T. B., & Tibshirani, R. J. (2014). `genlasso`: *Path Algorithm for Generalized Lasso Problems*. R package version 1.3.

- Arnold, T. B., & Tibshirani, R. J. (2016). Efficient implementations of the generalized lasso dual path algorithm. *Journal of Computational and Graphical Statistics*, *25*(1), 1–27.
- Bader, B. W., Kolda, T. G., et al. (2015). Matlab tensor toolbox version 2.6. Available online, <http://www.sandia.gov/tgkolda/TensorToolbox/>.
- Bar-Joseph, Z. (2004). Analyzing time series gene expression data. *Bioinformatics*, *20*(16), 2493–2503.
- Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms*. New York: Plenum Press.
- Bhar, A., Haubrock, M., Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S., & Wingender, E. (2013). Coexpression and coregulation analysis of time-series gene expression data in estrogen-induced breast cancer cell. *Algorithms for molecular biology*, *8*(1), 9.
- Bhar, A., Haubrock, M., Mukhopadhyay, A., & Wingender, E. (2015). Multiobjective triclustering of time-series transcriptome data reveals key genes of biological processes. *BMC bioinformatics*, *16*(1), 200.
- Björck, Å. (2015). *Numerical Methods in Matrix Computations*. New York, NY: Springer.
- Bontemps, D., Toussile, W., et al. (2013). Clustering and variable selection for categorical multivariate data. *Electronic Journal of Statistics*, *7*, 2344–2371.
- Boos, D., & Stefanski, L. (2013). *Essential Statistical Inference*. New York: Springer, 1st ed.
- Bouguila, N. (2010). On multivariate binary data clustering and feature weighting. *Computational Statistics & Data Analysis*, *54*(1), 120–134.

- Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1), 1–122.
- Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press.
- Braga Araújo, R., Trielli Ferreira, G. H., Orair, G. H., Meira, W., Celso Ferreira, R. A., Olavo Guedes Neto, D., & Zaki, M. J. (2008). The partricluster algorithm for gene expression analysis. *International Journal of Parallel Programming*, 36(2), 226–249.
- Bredel, M., Bredel, C., Juric, D., Harsh, G. R., Vogel, H., Recht, L. D., & Sikic, B. I. (2005). High-resolution genome-wide mapping of genetic alterations in human glial brain tumors. *Cancer Research*, 65(10), 4088–4096.
- Cao, X., Wei, X., Han, Y., & Lin, D. (2015). Robust face clustering via tensor decomposition. *IEEE transactions on cybernetics*, 45(11), 2546–2557.
- Carroll, J. D., & Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, 35(3), 283–319.
- Casella, G., & Berger, R. (2001). *Statistical Inference*. Pacific Grove, CA: Duxbury, 2nd ed.
- Chen, A. I., & Ozdaglar, A. (2012). A fast distributed proximal-gradient method. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, (pp. 601–608). IEEE.
- Chen, G. K., Chi, E. C., Ranola, J. M. O., & Lange, K. (2015). Convex clustering: an

- attractive alternative to hierarchical clustering. *PLoS computational biology*, 11(5), 1–31.
- Chen, J., & Chen, Z. (2008). Extended bayesian information criteria for model selection with large model spaces. *Biometrika*, 95(3), 759–771.
- Chen, J., & Chen, Z. (2012). Extended bic for small-n-large-p sparse glm. *Statistica Sinica*, (pp. 555–574).
- Chen, Z., & Luo, S. (2013). Selection consistency of ebic for glim with non-canonical links and diverging number of parameters. *Statistics and Its Interface*, 6(2), 275–284.
- Cheng, Y., & Church, G. M. (2000). Biclustering of expression data. In *In Proceedings of the 8th International Conference of Intelligent Systems and Molecular Biology*, vol. 8, (pp. 93–103).
- Chi, E. C., Allen, G. I., & Baraniuk, R. G. (2017). Convex biclustering. *Biometrics*, 73(1), 10–19.
- Chi, E. C., & Lange, K. (2015). Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, 24(4), 994–1013.
- Chrétien, S., Dombry, C., & Faivre, A. (2016). A semi-definite programming approach to low dimensional embedding for unsupervised clustering. *arXiv preprint arXiv:1606.09190*.
- Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., & Phan, H. A. (2015). Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2), 145–163.

- Combettes, P. L., & Pesquet, J.-C. (2011). Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, (pp. 185–212). Springer.
- Combettes, P. L., & Wajs, V. R. (2005). Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4), 1168–1200.
- Conesa, A., Nueda, M. J., Ferrer, A., & Talón, M. (2006). masigpro: a method to identify significantly differential expression profiles in time-course microarray experiments. *Bioinformatics*, 22(9), 1096–1102.
- De Lathauwer, L., De Moor, B., & Vandewalle, J. (2000). A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4), 1253–1278.
- Dede, D., & Oğul, H. (2013). A three-way clustering approach to cross-species gene regulation analysis. In *Innovations in Intelligent Systems and Applications (INISTA), 2013 IEEE International Symposium on*, (pp. 1–5). IEEE.
- Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 269–274). ACM.
- Ding, C., Li, T., Peng, W., & Park, H. (2006). Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 126–135). ACM.
- Ding, C., & Ye, J. (2005). *2-Dimensional Singular Value Decomposition for 2D Maps and Images*, (pp. 32–43).

- Dunn, J. C. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3), 32–57.
- Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, 32(2), 407–499.
- El-Arini, K., Xu, M., Fox, E. B., & Guestrin, C. (2013). Representing documents through their readers. In *Proceedings of the 19th Association for Computing Machinery International Conference on Knowledge Discovery and Data Mining*, (pp. 14–22).
- Foygel, R., & Drton, M. (2010). Extended bayesian information criteria for gaussian graphical models. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, & A. Culotta (Eds.) *Advances in Neural Information Processing Systems 23*, (pp. 604–612). Curran Associates, Inc.
- Friedman, J., Hastie, T., & Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3), 432–441.
- Geisser, S. (1975). The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350), 320–328.
- Gentle, J. E. (2007). *Matrix Algebra: Theory, Computations, and Applications in Statistics*. New York, NY: Springer.
- Goldstein, T., Studer, C., & Baraniuk, R. (2014). A field guide to forward-backward splitting with a FASTA implementation. *arXiv eprint*, [arXiv:abs/1411.3406](https://arxiv.org/abs/1411.3406).
- Goldstein, T., Studer, C., & Baraniuk, R. (2015). FASTA: A generalized implementation of forward-backward splitting. *arXiv eprint*, [arXiv:abs/1501.04979](https://arxiv.org/abs/1501.04979).

- Goodnight, J. H. (1979). A tutorial on the sweep operator. *The American Statistician*, 33(3), 149–158.
- Grantham, N. S. (2014). Clustering binary data with bernoulli mixture models. Unpublished written preliminary exam, NC State University.
- Guigourès, R., Boullé, M., & Rossi, F. (2015). Discovering patterns in time-varying graphs: a triclustering approach. *Advances in Data Analysis and Classification*, (pp. 1–28).
- Gutiérrez-Avilés, D., & Rubio-Escudero, C. (2014). Mining 3d patterns from gene expression temporal data: a new tricluster evaluation measure. *The Scientific World Journal*, 2014, 1–16.
- Gutiérrez-Avilés, D., Rubio-Escudero, C., Martínez-Álvarez, F., & Riquelme, J. C. (2014). Trigen: A genetic algorithm to mine triclusters in temporal gene expression data. *Neurocomputing*, 132, 42–53.
- Harshman, R. A. (1970). Foundations of the parafac procedure: models and conditions for an” explanatory” multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16, 1-84.
- Hašan, M., Velázquez-Armendáriz, E., Pellacini, F., & Bala, K. (2008). Tensor clustering for rendering many-light animations. In *Computer Graphics Forum*, vol. 27, (pp. 1105–1114). Wiley Online Library.
- Hasnat, M. A., Velcin, J., Bonnevey, S., & Jacques, J. (2017). Evolutionary clustering for categorical data using parametric links among multinomial mixture models. *Econometrics and Statistics*, 3, 141–159.

- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2 ed.
- He, T. (2011). *Lasso and General L1-Regularized Regression Under Linear Equality and Inequality Constraints*. Ph.D. thesis, Purdue University, Dept. of Statistics.
- He, X., Cai, D., Liu, H., & Han, J. (2005). Image clustering with tensor representation. In *Proceedings of the 13th annual ACM international conference on Multimedia*, (pp. 132–140). ACM.
- Hocking, T., Vert, J.-P., Bach, F., & Joulin, A. (2011). Clusterpath: an algorithm for clustering using convex fusion penalties. In L. Getoor, & T. Scheffer (Eds.) *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, (pp. 745–752). New York, NY, USA: ACM.
- Hoerl, A. E., & Kennard, R. W. (1970). Ridgerregression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1), 55–67.
- Hof, R. (2014). Study: Mobile ads actually do work - especially in apps. *Forbes*, August 27, 2014. Last Accessed July 9, 2017 from <https://www.forbes.com/sites/roberthof/2014/08/27/study-mobile-ads-actually-do-work-especially-in-apps/#27ce654057aa>.
- Holler, E., Butzhammer, P., Schmid, K., Hundsrucker, C., Koestler, J., Peter, K., Zhu, W., Sporrer, D., Hehlhans, T., Kreutz, M., Holler, B., Wolff, D., Edinger, M., Andreesen, R., Levine, J. E., Ferrara, J. L., Gessner, A., Spang, R., & Oefner, P. J. (2014). Metagenomic analysis of the stool microbiome in patients receiving allogeneic stem cell transplantation: Loss of diversity is associated with use of systemic antibi-

- otics and more pronounced in gastrointestinal graft-versus-host disease. *Biology of Blood and Marrow Transplantation*, 20(5), 640–645.
- Hopcroft, J., & Tarjan, R. (1973). Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6), 372–378.
- Houseman, E. A., Christensen, B. C., Yeh, R.-F., Marsit, C. J., Karagas, M. R., Wrensch, M., Nelson, H. H., Wiemels, J., Zheng, S., Wiencke, J. K., et al. (2008). Model-based clustering of dna methylation array data: a recursive-partitioning algorithm for high-dimensional data arising as a mixture of beta distributions. *BMC bioinformatics*, 9(1), 1–15.
- Hu, Q., Zeng, P., & Lin, L. (2015a). The dual and degrees of freedom of linearly constrained generalized lasso. *Computational Statistics & Data Analysis*, 86, 13–26.
- Hu, Z., Follmann, D. A., & Miura, K. (2015b). Vaccine design via nonnegative lasso-based variable selection. *Statistics in Medicine*, 34(10), 1791–1798.
- Huang, H., Ding, C., Luo, D., & Li, T. (2008). Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, (pp. 327–335). ACM.
- Huang, H., Yan, J., Nie, F., Huang, J., Cai, W., Saykin, A. J., & Shen, L. (2013a). A new sparse simplex model for brain anatomical and genetic network analysis. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, (pp. 625–632).
- Huang, T., Gong, H., Yang, C., & He, Z. (2013b). Proteinlasso: A lasso regression

- approach to protein inference problem in shotgun proteomics. *Computational Biology and Chemistry*, 43, 46–54.
- Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, 2(3), 283–304.
- Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of classification*, 2(1), 193–218.
- Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, (pp. 604–613). ACM.
- Jain, P., & Oh, S. (2014). Provable tensor factorization with missing data. In *Advances in Neural Information Processing Systems*, (pp. 1431–1439).
- James, G. M., Paulson, C., & Rusmevichientong, P. (2013). Penalized and constrained regression. unpublished manuscript, University of Southern California.
- Jegelka, S., Sra, S., & Banerjee, A. (2009). Approximation algorithms for tensor clustering. In *International Conference on Algorithmic Learning Theory*, (pp. 368–383). Springer.
- Jiang, H., Zhou, S., Guan, J., Zheng, Y., et al. (2006). gtricluster: a more general and effective 3d clustering algorithm for gene-sample-time microarray data. *BioDM*, 6, 48–59.
- Jones, P., Parker, D., Osborn, T., & Briffa, K. (2016). Global and hemispheric temperature anomalies - land and marine instrumental records. *Trends: A Compendium of Data on Global Change*.

- Kakati, T., Ahmed, H. A., Bhattacharyya, D. K., & Kalita, J. K. (2016). A fast gene expression analysis using parallel biclustering and distributed triclustering approach. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, (p. 122). ACM.
- Kim, S.-J., Koh, K., Boyd, S., & Gorinevsky, D. (2009). ℓ_1 trend filtering. *SIAM review*, *51*(2), 339–360.
- Koestler, D. C., Christensen, B. C., Marsit, C. J., Kelsey, K. T., & Houseman, E. A. (2013). Recursively partitioned mixture model clustering of dna methylation data using biologically informed correlation structures. *Statistical applications in genetics and molecular biology*, *12*(2), 225–240.
- Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, *51*(3), 455–500.
- Kolda, T. G., Bader, B. W., & Kenny, J. P. (2005). Higher-order web link analysis using multilinear algebra. In *Data Mining, Fifth IEEE International Conference on*, (pp. 242–249). IEEE.
- Kolda, T. G., & Sun, J. (2008). Scalable tensor decompositions for multi-aspect data mining. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, (pp. 363–372). IEEE.
- Kump, P., Bai, E.-W., Chan, K.-S., Eichinger, B., & Li, K. (2012). Variable selection via rival (removing irrelevant variables amidst lasso iterations) and its application to nuclear material detection. *Automatica*, *48*(9), 2107–2115.

- Kutty, S., Nayak, R., & Li, Y. (2011). Xml documents clustering using a tensor space model. *Advances in Knowledge Discovery and Data Mining*, (pp. 488–499).
- Lange, K. (2013). *Optimization*. New York: Springer, 2nd ed.
- Lange, K. (2016). *MM Optimization Algorithms*. SIAM.
- Li, H. (2015). Microbiome, metagenomics, and high-dimensional compositional data analysis. *Annual Review of Statistics and Its Application*, 2, 73–94.
- Li, J., & Zha, H. (2006). Two-way poisson mixture models for simultaneous document classification and word clustering. *Computational Statistics & Data Analysis*, 50(1), 163–180.
- Li, M., Andersen, D. G., & Smola, A. (2013). Distributed delayed proximal gradient methods. In *NIPS Workshop on Optimization for Machine Learning*, vol. 3, (pp. 1–5).
- Li, T. (2006). A unified view on clustering binary data. *Machine Learning*, 62(3), 199–215.
- Li, T., & Zhu, S. (2005). On clustering binary data. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, (pp. 526–530). SIAM.
- Lichman, M. (2013). UCI machine learning repository. Available online, <http://archive.ics.uci.edu/ml>.
- Lin, W., Shi, P., Feng, R., & Li, H. (2014). Variable selection in regression with compositional covariates. *Biometrika*, 101(4), 785–797.

- Lindsten, F., Ohlsson, H., & Ljung, L. (2011). Clustering using sum-of-norms regularization: With application to particle filter output computation. In *Statistical Signal Processing Workshop (SSP), 2011 IEEE*, (pp. 201–204). IEEE.
- Liu, X., Ji, S., Glänzel, W., & De Moor, B. (2013). Multiview partitioning via tensor methods. *IEEE Transactions on Knowledge and Data Engineering*, 25(5), 1056–1069.
- Liu, Y., Yang, T., & Fu, L. (2015). A partitioning based algorithm to fuzzy tricluster. *Mathematical Problems in Engineering*, 2015.
- Luo, S., & Chen, Z. (2013). Extended bic for linear regression models with diverging number of relevant features and high or ultra-high feature spaces. *Journal of Statistical Planning and Inference*, 143(3), 494–504.
- Luo, S., & Chen, Z. (2014). Sequential lasso cum ebic for feature selection with ultra-high dimensional feature space. *Journal of the American Statistical Association*, 109(507), 1229–1240.
- Luo, S., Xu, J., & Chen, Z. (2015). Extended bayesian information criterion in the cox model with a high-dimensional feature space. *Annals of the Institute of Statistical Mathematics*, 67(2), 287–311.
- Ma, Z., Teschendorff, A. E., Yu, H., Taghia, J., & Guo, J. (2014). Comparisons of non-gaussian statistical models in dna methylation analysis. *International journal of molecular sciences*, 15(6), 10835–10854.
- Madeira, S. C., & Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: a survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 1(1), 24–45.

- Mallows, C. L. (1973). Some comments on c p. *Technometrics*, 15(4), 661–675.
- Mampaey, M., & Vreeken, J. (2013). Summarizing categorical data by clustering attributes. *Data Mining and Knowledge Discovery*, (pp. 1–44).
- Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.
- McCullagh, P., & Nelder, J. A. (1989). *Generalized Linear Models*. London.
- McGarry, C. (2016). Report: Google is the default iphone search engine because it paid apple \$1 billion. *Macworld*, January 22, 2016. Last Accessed July 9, 2017 from <http://www.macworld.com/article/3025783/iphone-ipad/report-google-is-the-default-iphone-search-engine-because-it-paid-apple-1-billion.html>.
- Meilă, M. (2007). Comparing clusterings an information based distance. *Journal of multivariate analysis*, 98(5), 873–895.
- Meinshausen, N., & Bühlmann, P. (2010). Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4), 417–473.
- Metzler, S., & Miettinen, P. (2015). Clustering boolean tensors. *Data Mining and Knowledge Discovery*, 29(5), 1343–1373.
- Michels, E., De Preter, K., Van Roy, N., & Speleman, F. (2007). Detection of dna copy number alterations in cancer by array comparative genomic hybridization. *Genetics in Medicine*, 9(9), 574–584.
- Mitchell, A., Rosenstiel, T., Santhanam, L. H., & Christian, L. (2012). Future of mobile news. *Project for Excellence in Journalism (PEJ)— Understanding News in the Information Age*.

- Muja, M., & Lowe, D. G. (2009). Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application (VISSAPP 2009)*, (pp. 331–340). INSTICC Press.
- Murphy, S., & Nguyen, V. H. (2011). Role of gut microbiota in graft-versus-host disease. *Leukemia & Lymphoma*, *52*(10), 1844–1856.
- Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, (pp. 849–856).
- Oh, J., Shin, K., Papalexakis, E. E., Faloutsos, C., & Yu, H. (2017). S-hot: Scalable high-order tucker decomposition. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, (pp. 761–770). ACM.
- O’Sullivan, F. (1986). A statistical perspective on ill-posed inverse problems. *Statistical Science*, (pp. 502–518).
- Papalexakis, E. E., Faloutsos, C., & Sidiropoulos, N. D. (2016). Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *8*(2), 1–44.
- Papalexakis, E. E., Sidiropoulos, N. D., & Bro, R. (2013). From k-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE transactions on signal processing*, *61*(2), 493–506.
- Parikh, N., Boyd, S., et al. (2014). Proximal algorithms. *Foundations and Trends® in Optimization*, *1*(3), 127–239.
- Peng, W., & Li, T. (2011). Tensor clustering via adaptive subspace iteration. *Intelligent Data Analysis*, *15*(5), 695–713.

- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14(5), 465–471.
- Rosset, S., & Zhu, J. (2007). Piecewise linear regularized solution paths. *The Annals of Statistics*, 35(3), 1012–1030.
- Schwarz, G., et al. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2), 461–464.
- Shi, P., Zhang, A., & Li, H. (2016). Regression analysis for microbiome compositional data. *Annals of Applied Statistics*, 10(2), 1019–1040.
- Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., & Faloutsos, C. (2016). Tensor decomposition for signal processing and machine learning. *arXiv preprint arXiv:1607.01668*.
- Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics*, 9(6), 1135–1151.
- Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society. Series B (Methodological)*, (pp. 111–147).
- Sun, J., Papadimitriou, S., Lin, C.-Y., Cao, N., Liu, S., & Qian, W. (2009). Multivis: Content-based social network exploration through multi-way visual analysis. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, (pp. 1064–1075). SIAM.
- Sun, J., Tao, D., & Faloutsos, C. (2006). Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 374–383). ACM.

- Sun, W. W., Lu, J., Liu, H., & Cheng, G. (2015). Provable sparse tensor decomposition. *arXiv preprint arXiv:1502.01425*.
- Sun, Y., Gao, J., Hong, X., Guo, Y., & Harris, C. J. (2014). Dimensionality reduction assisted tensor clustering. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, (pp. 1565–1572). IEEE.
- Sun, Y., Gao, J., Hong, X., Mishra, B., & Yin, B. (2016). Heterogeneous tensor decomposition for clustering via manifold optimization. *IEEE transactions on pattern analysis and machine intelligence*, 38(3), 476–489.
- Tamhane, A. C., Qiu, D., & Ankenman, B. E. (2010). A parametric mixture model for clustering multivariate binary data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 3(1), 3–19.
- Tan, K. M., & Witten, D. (2015). Statistical properties of convex clustering. *Electronic journal of statistics*, 9(2), 2324.
- Tan, K. M., & Witten, D. M. (2014). Sparse biclustering of transposable data. *Journal of Computational and Graphical Statistics*, 23(4), 985–1008.
- Tang, J., Shu, X., Qi, G.-J., Li, Z., Wang, M., Yan, S., & Jain, R. (2017). Tri-clustered tensor completion for social-aware image tag refinement. *IEEE transactions on pattern analysis and machine intelligence*, 39(8), 1662–1674.
- Taur, Y., Xavier, J. B., Lipuma, L., Ubeda, C., Goldberg, J., Gobourne, A., Lee, Y. J., Dubin, K. A., Socci, N. D., Viale, A., Perales, M.-A., Jenq, R. R., van den Brink, M. R. M., & Pamer, E. G. (2012). Intestinal domination and the risk of bacteremia

- in patients undergoing allogeneic hematopoietic stem cell transplantation. *Clinical Infectious Diseases*, 55(7), 905–914.
- Tchagang, A. B., Phan, S., Famili, F., Shearer, H., Fobert, P., Huang, Y., Zou, J., Huang, D., Cutler, A., Liu, Z., et al. (2012). Mining biological information from 3d short time-series gene expression data: the optricluster algorithm. *BMC bioinformatics*, 13(1), 54.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267–288.
- Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., & Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1), 91–108.
- Tibshirani, R., & Suo, X. (2016). An ordered lasso and sparse time-lagged regression. *Technometrics*, 58(4), 415–423.
- Tibshirani, R., Walther, G., & Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2), 411–423.
- Tibshirani, R., & Wang, P. (2008). Spatial smoothing and hot spot detection for cgh data using the fused lasso. *Biostatistics*, 9(1), 18–29.
- Tibshirani, R. J., Hoefling, H., & Tibshirani, R. (2011). Nearly-isotonic regression. *Technometrics*, 53(1), 54–61.
- Tibshirani, R. J., & Taylor, J. (2011). The solution path of the generalized lasso. *The Annals of Statistics*, 39(3), 1335–1371.

- Tibshirani, R. J., & Taylor, J. (2012). Degrees of freedom in lasso problems. *The Annals of Statistics*, *40*(2), 1198–1232.
- Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, *31*(3), 279–311.
- Vervliet, N., Debals, O., Sorber, L., Van Barel, M., & De Lathauwer, L. (2016). Tensorlab 3.0. Available online, <http://www.tensorlab.net>.
- Vichi, M., Rocci, R., & Kiers, H. A. (2007). Simultaneous component and clustering models for three-way data: within and between approaches. *Journal of Classification*, *24*(1), 71–98.
- Wang, B., Zhang, Y., Sun, W. W., & Fang, Y. (2017). Sparse convex clustering. *arXiv preprint arXiv:1601.04586*.
- Wang, G., Yin, L., Zhao, Y., & Mao, K. (2010). Efficiently mining time-delayed gene expression patterns. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *40*(2), 400–411.
- Weber, D., Oefner, P. J., Hiergeist, A., Koestler, J., Gessner, A., Weber, M., Hahn, J., Wolff, D., Stämmeler, F., Spang, R., Herr, W., Dettmer, K., & Holler, E. (2015). Low urinary undoxyl sulfate levels early after transplantation reflect a disrupted microbiome and are associated with poor outcome. *Blood*, *126*(14), 1723–1728.
- Witten, D. M. (2011). Classification and clustering of sequencing data using a poisson model. *The Annals of Applied Statistics*, *5*(4), 2493–2518.
- Wu, C., Yang, H., Zhu, J., Zhang, J., King, I., & Lyu, M. R. (2013). Sparse poisson coding

- for high dimensional document clustering. In *Big Data, 2013 IEEE International Conference on*, (pp. 512–517). IEEE.
- Wu, L., Yang, Y., & Liu, H. (2014). Nonnegative-lasso and application in index tracking. *Computational Statistics & Data Analysis*, *70*, 116–126.
- Wu, T., Benson, A. R., & Gleich, D. F. (2016). General tensor spectral co-clustering for higher-order data. In *Advances in Neural Information Processing Systems*, (pp. 2559–2567).
- Wu, W. B., Woodroffe, M., & Mentz, G. (2001). Isotonic regression: Another look at the changepoint problem. *Biometrika*, *88*(3), 793–804.
- Xu, X., Lu, Y., Tan, K.-L., & Tung, A. K. (2009). Finding time-lagged 3d clusters. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, (pp. 445–456). IEEE.
- Yokota, T., Lee, N., & Cichocki, A. (2017). Robust multilinear tensor rank estimation using higher order singular value decomposition and information criteria. *IEEE Transactions on Signal Processing*, *65*(5), 1196–1206.
- Yu, H., Luscombe, N. M., Qian, J., & Gerstein, M. (2003). Genomic analysis of gene expression relationships in transcriptional regulatory networks. *TRENDS in Genetics*, *19*(8), 422–427.
- Yuan, M., & Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *68*(1), 49–67.

- Yuan, M., & Lin, Y. (2007). Model selection and estimation in the gaussian graphical model. *Biometrika*, *94*(1), 19–35.
- Zhang, Z.-Y., Li, T., & Ding, C. (2013). Non-negative tri-factor tensor decomposition with applications. *Knowledge and information systems*, *34*(2), 243–265.
- Zhao, J., Xie, X., Xu, X., & Sun, S. (2017). Multi-view learning overview: Recent progress and new challenges. *Information Fusion*, *38*, 43–54.
- Zhao, L., & Zaki, M. J. (2005). Tricluster: an effective algorithm for mining coherent clusters in 3d microarray data. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, (pp. 694–705). ACM.
- Zhou, H., & Lange, K. (2009). Rating movies and rating the raters who rate them. *The American Statistician*, *63*(4), 297–307.
- Zhou, H., & Lange, K. (2013). A path algorithm for constrained estimation. *Journal of Computational and Graphical Statistics*, *22*(2), 261–283.
- Zhou, H., & Wu, Y. (2014). A generic path algorithm for regularized statistical estimation. *Journal of the American Statistical Association*, *109*(506), 686–699.
- Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, *101*(476), 1418–1429.
- Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *67*(2), 301–320.
- Zou, H., Hastie, T., & Tibshirani, R. (2007). On the degrees of freedom of the lasso. *The Annals of Statistics*, *35*(5), 2173–2192.

APPENDICES

Appendix A

Additional Constrained Lasso

Derivations

A.1 Constrained Lasso via Generalized Lasso

As detailed in Section 2.2, it is always possible to reformulate and solve a generalized lasso as a constrained lasso. In this section, we demonstrate that it is not always possible to transform a constrained lasso to a generalized lasso. However, we first examine a situation where it is in fact possible to transform a constrained lasso to a generalized lasso.

A.1.1 Reparameterization

Consider a constrained lasso with only equality constraints and $\mathbf{b} = \mathbf{0}_q$,

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho \|\boldsymbol{\beta}\|_1 && \text{(A.1)} \\ & \text{subject to} && \mathbf{A}\boldsymbol{\beta} = \mathbf{0}_q, \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{q \times p}$ with $\text{rank}(\mathbf{A}) = q$. Consider a matrix $\mathbf{D} \in \mathbb{R}^{p \times p-q}$ whose columns span the null space of \mathbf{A} . For example, we can use \mathbf{Q}_2 from the QR decomposition of \mathbf{A}^T . Then we can use the change of variables

$$\boldsymbol{\beta} = \mathbf{D}\boldsymbol{\theta},$$

so the objective function becomes

$$\text{minimize} \quad \frac{1}{2} \|\mathbf{y} - \mathbf{X}\mathbf{D}\boldsymbol{\theta}\|_2^2 + \rho \|\mathbf{D}\boldsymbol{\theta}\|_1, \quad (\text{A.2})$$

and the constraints can be written as

$$\mathbf{A}\boldsymbol{\beta} = \mathbf{A}\mathbf{D}\boldsymbol{\theta} = \mathbf{0}\boldsymbol{\theta} = \mathbf{0}_q.$$

Thus the constraints vanish, as they hold for all $\boldsymbol{\theta}$, and we are left with an unconstrained generalized lasso (A.2). This result is not surprising in light of the result in Section 2.2, which showed that a generalized lasso reformulated as a constrained lasso has the constraints $\mathbf{U}_2^T \boldsymbol{\alpha} = \mathbf{0}_{m-r}$. That is a case where $\mathbf{b} = \mathbf{0}_q$ and the resulting constrained lasso solution can be translated back to the original generalized lasso parameterization via an affine transformation, in line with the result in this section that this is a situation where the constrained lasso can in fact be transformed to a generalized lasso.

Now consider the more general case with an arbitrary $\mathbf{b} \neq \mathbf{0}_q$. We can re-arrange the

equality constraints as

$$\begin{aligned} \mathbf{A}\boldsymbol{\beta} &= \mathbf{b} \\ \mathbf{A}\boldsymbol{\beta} - \mathbf{b} &= \mathbf{0}_q \\ \begin{pmatrix} \mathbf{A}, & -\mathbf{I}_q \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta} \\ \mathbf{b} \end{pmatrix} &= \tilde{\mathbf{A}}\tilde{\boldsymbol{\beta}} = \mathbf{0}_q, \end{aligned}$$

and then apply the above result using $\tilde{\mathbf{D}} = \tilde{\mathbf{Q}}_2$ from the QR decomposition of $\tilde{\mathbf{A}}^T$. However, now the reparameterized problem is

$$\tilde{\boldsymbol{\beta}} = \tilde{\mathbf{D}}\tilde{\boldsymbol{\theta}} \begin{pmatrix} \tilde{\mathbf{D}}_1\tilde{\boldsymbol{\theta}}_1 \\ \tilde{\mathbf{D}}_2\tilde{\boldsymbol{\theta}}_2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\beta} \\ \mathbf{b} \end{pmatrix},$$

we are still left with the constraint $\tilde{\mathbf{D}}_2\tilde{\boldsymbol{\theta}}_2 = \mathbf{b}$. Therefore, for equality constraints with $\mathbf{b} \neq \mathbf{0}_q$, a constrained lasso can be transformed into a constrained generalized lasso. This result is trivial, however, since a constrained lasso is always a constrained generalized lasso with $\mathbf{D} = \mathbf{I}_p$.

A.1.2 Null-Space Method

Another common method for solving least squares problems with equality constraints (LSE) is the null-space method (Björck, 2015). To apply this method to the constrained lasso, we again restrict our attention to a constrained lasso with only equality constraints for the time being,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \rho\|\boldsymbol{\beta}\|_1 & (\text{A.3}) \\ \text{subject to} \quad & \mathbf{A}\boldsymbol{\beta} = \mathbf{b}. \end{aligned}$$

We will show that the application of the null-space method results in a shifted generalized lasso. Consider the QR decomposition of $\mathbf{A}^T \in \mathbb{R}^{p \times q}$, where $\text{rank}(\mathbf{A}) = q$,

$$\mathbf{A}^T = \mathbf{Q} \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} \text{ with } \mathbf{Q}\mathbf{Q}' = \mathbf{I}_p,$$

and $\mathbf{Q} \in \mathbb{R}^{p \times p}$, $\mathbf{R} \in \mathbb{R}^{q \times q}$, and $\mathbf{0} \in \mathbb{R}^{p-q \times q}$. Let

$$\mathbf{X}\mathbf{Q} = \begin{pmatrix} \mathbf{X}_1 & \mathbf{X}_2 \end{pmatrix} \text{ and } \mathbf{Q}'\boldsymbol{\beta} = \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{pmatrix},$$

with $\mathbf{X}_1 \in \mathbb{R}^{n \times q}$, $\mathbf{X}_2 \in \mathbb{R}^{n \times p-q}$, $\boldsymbol{\alpha}_1 \in \mathbb{R}^{q \times 1}$, and $\boldsymbol{\alpha}_2 \in \mathbb{R}^{p-q \times 1}$, then

$$\mathbf{X}\boldsymbol{\beta} = \mathbf{X}\mathbf{Q}\mathbf{Q}'\boldsymbol{\beta} = \begin{pmatrix} \mathbf{X}_1 & \mathbf{X}_2 \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{pmatrix} = \mathbf{X}_1\boldsymbol{\alpha}_1 + \mathbf{X}_2\boldsymbol{\alpha}_2.$$

As for the constraints, we have

$$\mathbf{A} = \begin{pmatrix} \mathbf{R}^T & \mathbf{0}^T \end{pmatrix} \mathbf{Q}^T \Rightarrow \mathbf{A}\boldsymbol{\beta} = \begin{pmatrix} \mathbf{R}^T & \mathbf{0}^T \end{pmatrix} \mathbf{Q}^T\boldsymbol{\beta} = \begin{pmatrix} \mathbf{R}^T & \mathbf{0}^T \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{pmatrix} = \mathbf{R}^T\boldsymbol{\alpha}_1.$$

Lastly, the penalty term becomes

$$\|\boldsymbol{\beta}\|_1 = \left\| \mathbf{Q} \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \end{pmatrix} \right\|_1 = \|\mathbf{Q}_1\boldsymbol{\alpha}_1 + \mathbf{Q}_2\boldsymbol{\alpha}_2\|_1.$$

Thus, putting these pieces together we can re-write (A.3) as

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|(\mathbf{y} - \mathbf{X}_1 \boldsymbol{\alpha}_1) - \mathbf{X}_2 \boldsymbol{\alpha}_2\|_2^2 + \rho \|\mathbf{Q}_1 \boldsymbol{\alpha}_1 + \mathbf{Q}_2 \boldsymbol{\alpha}_2\|_1 && (\text{A.4}) \\ & \text{subject to} && \mathbf{R}^T \boldsymbol{\alpha}_1 = \mathbf{b}. \end{aligned}$$

Since \mathbf{R}^T is invertible, we can further directly incorporate the constraints in the objective function by plugging in $\boldsymbol{\alpha}_1 = (\mathbf{R}^T)^{-1} \mathbf{b}$,

$$\text{minimize} \quad \frac{1}{2} \|(\mathbf{y} - \mathbf{X}_1 (\mathbf{R}^T)^{-1} \mathbf{b}) - \mathbf{X}_2 \boldsymbol{\alpha}_2\|_2^2 + \rho \|\mathbf{Q}_1 (\mathbf{R}^T)^{-1} \mathbf{b} + \mathbf{Q}_2 \boldsymbol{\alpha}_2\|_1, \quad (\text{A.5})$$

or more concisely as

$$\text{minimize} \quad \frac{1}{2} \|\tilde{\mathbf{y}} - \mathbf{X}_2 \boldsymbol{\alpha}_2\|_2^2 + \rho \|\mathbf{c} + \mathbf{Q}_2 \boldsymbol{\alpha}_2\|_1, \quad (\text{A.6})$$

with $\tilde{\mathbf{y}} = \mathbf{y} - \mathbf{X}_1 (\mathbf{R}^T)^{-1} \mathbf{b}$ and $\mathbf{c} = \mathbf{Q}_1 (\mathbf{R}^T)^{-1} \mathbf{b}$. So (A.6) resembles an unconstrained generalized lasso problem with $\mathbf{D} = \mathbf{Q}_2$, but it has been shifted by a constant vector $\mathbf{c} = \mathbf{Q}_1 (\mathbf{R}^T)^{-1} \mathbf{b}$ that can not be decoupled from the penalty term. Therefore, it once again is not possible to solve a constrained lasso by reformulating it as an unconstrained generalized lasso. It should be noted that such a transformation is not possible even in the presence of only equality constraints, and the addition of inequality constraints would only further complicate matters. As pointed out by Gentle (2007), there is no general closed-form solution to least squares problems with inequality constraints.

A.2 Subgradient Violations

As pointed out in Section 2.3.3, there is the potential for the subgradient conditions (2.8) to become violated if an inactive coefficient is moving too slowly. Here we provide

the supporting derivations for this result and what is meant by too slowly. To preview the result, an inactive coefficient, $j \in \mathcal{A}^c$, with subgradient $s_j = \pm 1$ is moved to the active set if $s_j \cdot \frac{d}{d\rho}[\rho s_j] < 1$. To see this, without loss of generality assume that $s_j = -1$ for some inactive coefficient β_j , $j \in \mathcal{A}^c$. As given in Table 2.1, since ρ is decreasing, s_j is updated along the path via

$$[\rho^{(t+1)} s_j^{(t+1)}] = \rho^{(t)} s_j^{(t)} - \Delta\rho \cdot \frac{d}{d\rho}[\rho s_j],$$

which implies

$$s_j^{(t+1)} = \left(\rho^{(t)} s_j^{(t)} - \Delta\rho \cdot \frac{d}{d\rho}[\rho s_j] \right) / \rho^{(t+1)}. \quad (\text{A.7})$$

Since ρ is decreasing, $\rho^{(t)} > \rho^{(t+1)}$, but we define $\Delta\rho > 0$ which implies that $\Delta\rho = \rho^{(t)} - \rho^{(t+1)}$. Using this and $s_j^{(t)} = -1$, then for a given inactive coefficient $j \in \mathcal{A}^c$, (A.7) becomes

$$s_j^{(t+1)} = \left(-\rho^{(t)} - (\rho^{(t)} - \rho^{(t+1)}) \frac{d}{d\rho}[\rho s_j] \right) / \rho^{(t+1)}. \quad (\text{A.8})$$

To identify the trouble ranges for $\frac{d}{d\rho}[\rho s_j]$ that would result in a violation of the subgradient conditions, we can rearrange (A.8) as follows,

$$\begin{aligned}
s_j^{(t+1)} &= \left(-\rho^{(t)} - (\rho^{(t)} - \rho^{(t+1)}) \frac{d}{d\rho}[\rho s_j] \right) / \rho^{(t+1)} \\
&= -\frac{d}{d\rho}[\rho s_j] \left(\frac{\rho^{(t)}}{\rho^{(t+1)}} - 1 \right) - \frac{\rho^{(t)}}{\rho^{(t+1)}} \\
&= -\frac{d}{d\rho}[\rho s_j] \left(\frac{\rho^{(t)}}{\rho^{(t+1)}} - 1 \right) - \frac{\rho^{(t)}}{\rho^{(t+1)}} + 1 - 1 \\
&= -\frac{d}{d\rho}[\rho s_j] \left(\frac{\rho^{(t)}}{\rho^{(t+1)}} - 1 \right) + \left(1 - \frac{\rho^{(t)}}{\rho^{(t+1)}} \right) - 1 \\
&= \left(\frac{d}{d\rho}[\rho s_j] + 1 \right) \left(1 - \frac{\rho^{(t)}}{\rho^{(t+1)}} \right) - 1. \tag{A.9}
\end{aligned}$$

The second term in the product in (A.9) is always negative, since $\rho^{(t)} > \rho^{(t+1)} \Rightarrow \rho^{(t)}/\rho^{(t+1)} > 1 \Rightarrow 0 > 1 - (\rho^{(t)}/\rho^{(t+1)})$. Now, consider different values for $\frac{d}{d\rho}[\rho s_j]$:

- i) $\frac{d}{d\rho}[\rho s_j] > -1$: When $\frac{d}{d\rho}[\rho s_j] > -1$, then $\left(\frac{d}{d\rho}[\rho s_j] + 1 \right) > 0$, so the product term in (A.9) involves a positive number multiplied by a negative number and is thus negative. However, this would lead to $s_j < -1$ when 1 is subtracted from the product term, which is a violation of the subgradient conditions.
- ii) $\frac{d}{d\rho}[\rho s_j] = -1$: This is fine as it maintains $s_j = -1$, since $\frac{d}{d\rho}[\rho s_j] = -1 \Rightarrow \left(\frac{d}{d\rho}[\rho s_j] + 1 \right) = 0 \Rightarrow s_j^{(t+1)} = -1$.
- iii) $\frac{d}{d\rho}[\rho s_j] < -1$: This situation is also fine as $\frac{d}{d\rho}[\rho s_j] < -1 \Rightarrow \left(\frac{d}{d\rho}[\rho s_j] + 1 \right) < 0$, so the product term in (A.9) is positive and the subgradient is moving towards zero, which is fine since $j \in \mathcal{A}^c$.

The only issue, then, arises when $\frac{d}{d\rho}[\rho s_j] > -1$. The corresponding range for $j \in \mathcal{A}^c$ but $s_j = 1$ is $\frac{d}{d\rho}[\rho s_j] < -1$, which is derived similarly. Combining these two situations, the

range to monitor can be written more succinctly as $s_j \cdot \frac{d}{d\rho}[\rho s_j] < 1$. Thus, to summarize, an inactive coefficient $j \in \mathcal{A}^c$ with subgradient $s_j = \pm 1$ and $s_j \cdot \frac{d}{d\rho}[\rho s_j] < 1$ needs to be moved back into the active set, \mathcal{A} , before the path algorithm proceeds to prevent a violation of the subgradient conditions (2.8).

Appendix B

Additional Convex Triclustering Simulation Results

B.1 Checkbox Pattern: Balanced Sizes and Homoskedasticity

$I_1 = I_2 = I_3 = 40$, Adjusted Rand Index

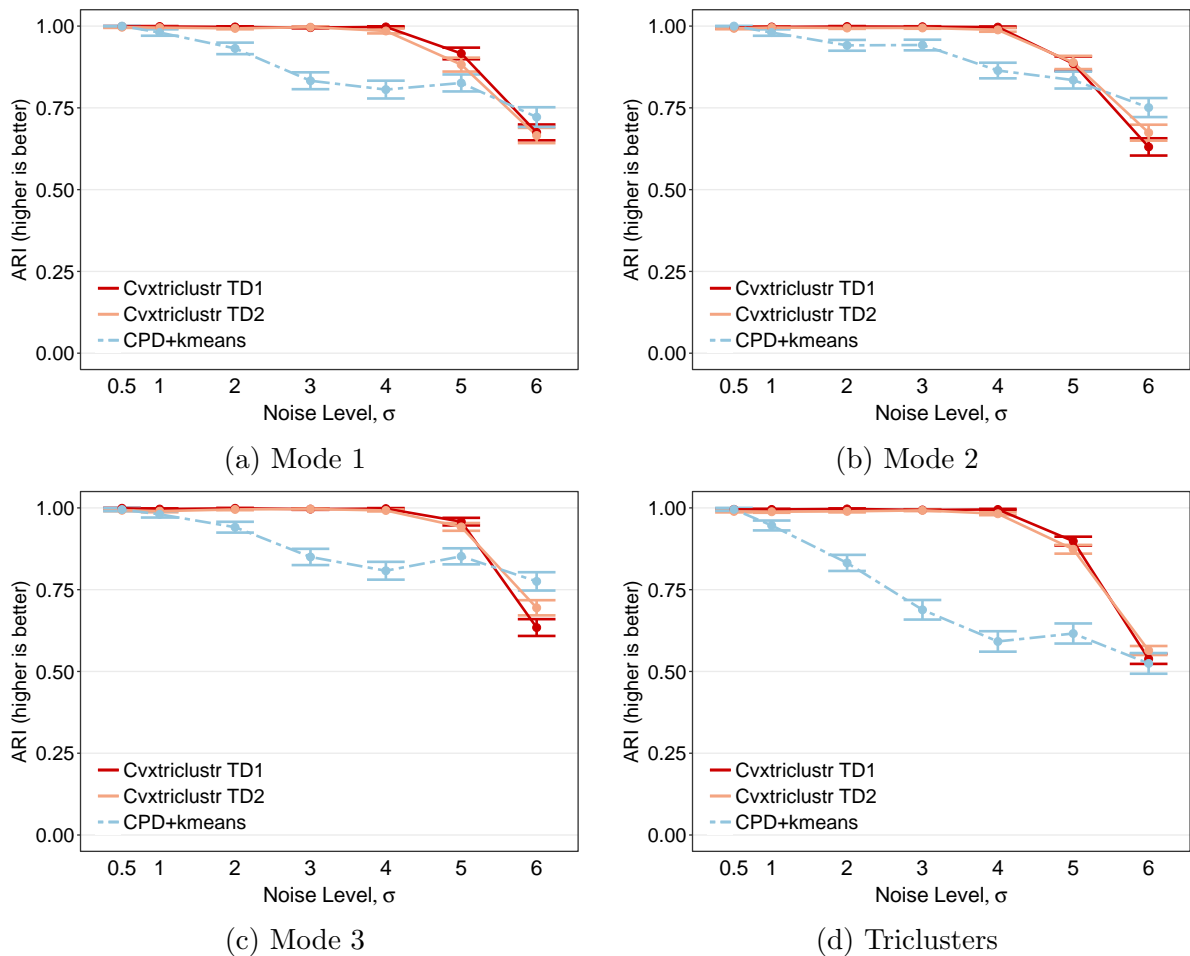


Figure B.1: Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average adjusted rand index plus/minus one standard error.

$I_1 = I_2 = I_3 = 40$, Variation of Information

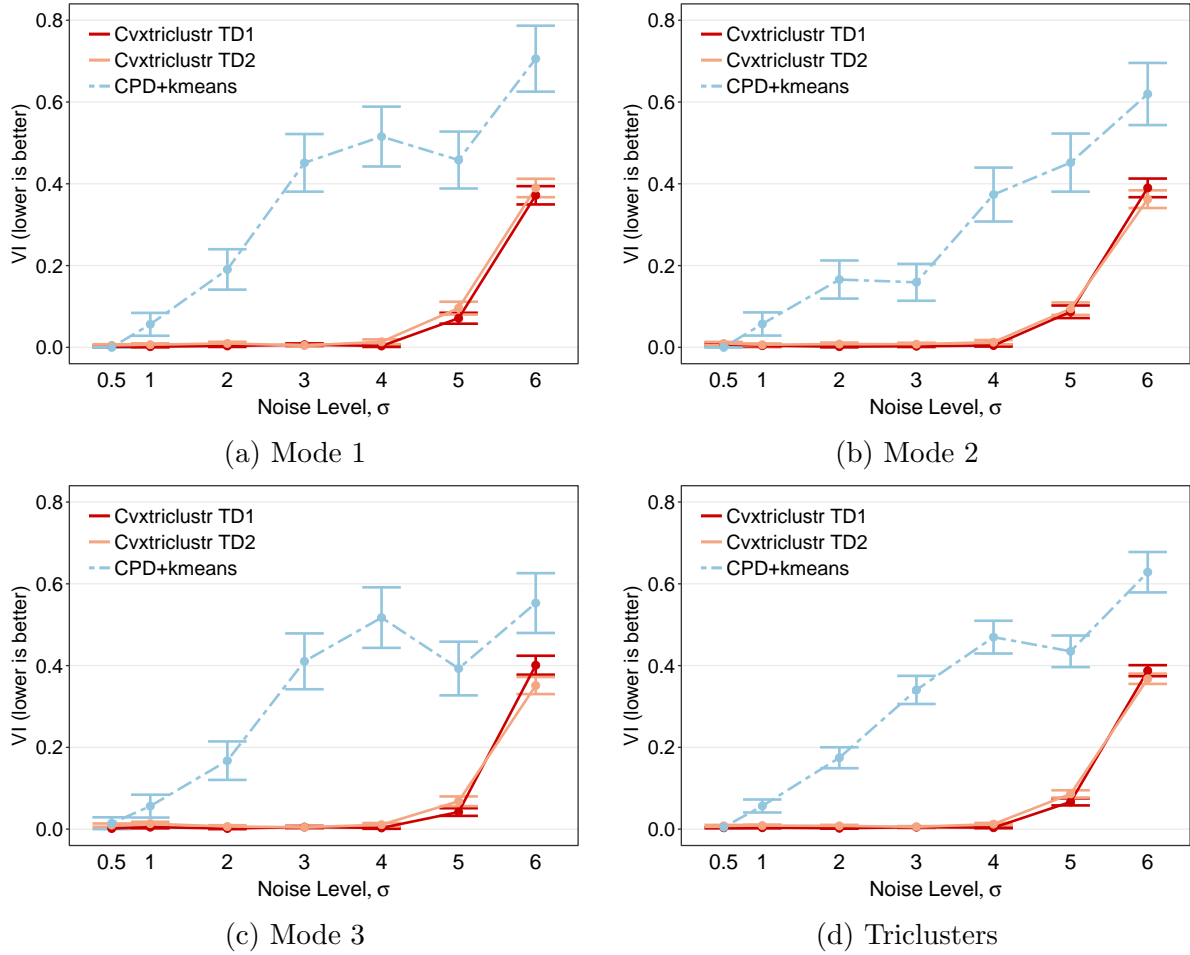


Figure B.2: Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average variation of information plus/minus one standard error.

$I_1 = I_2 = I_3 = 60$, Adjusted Rand Index

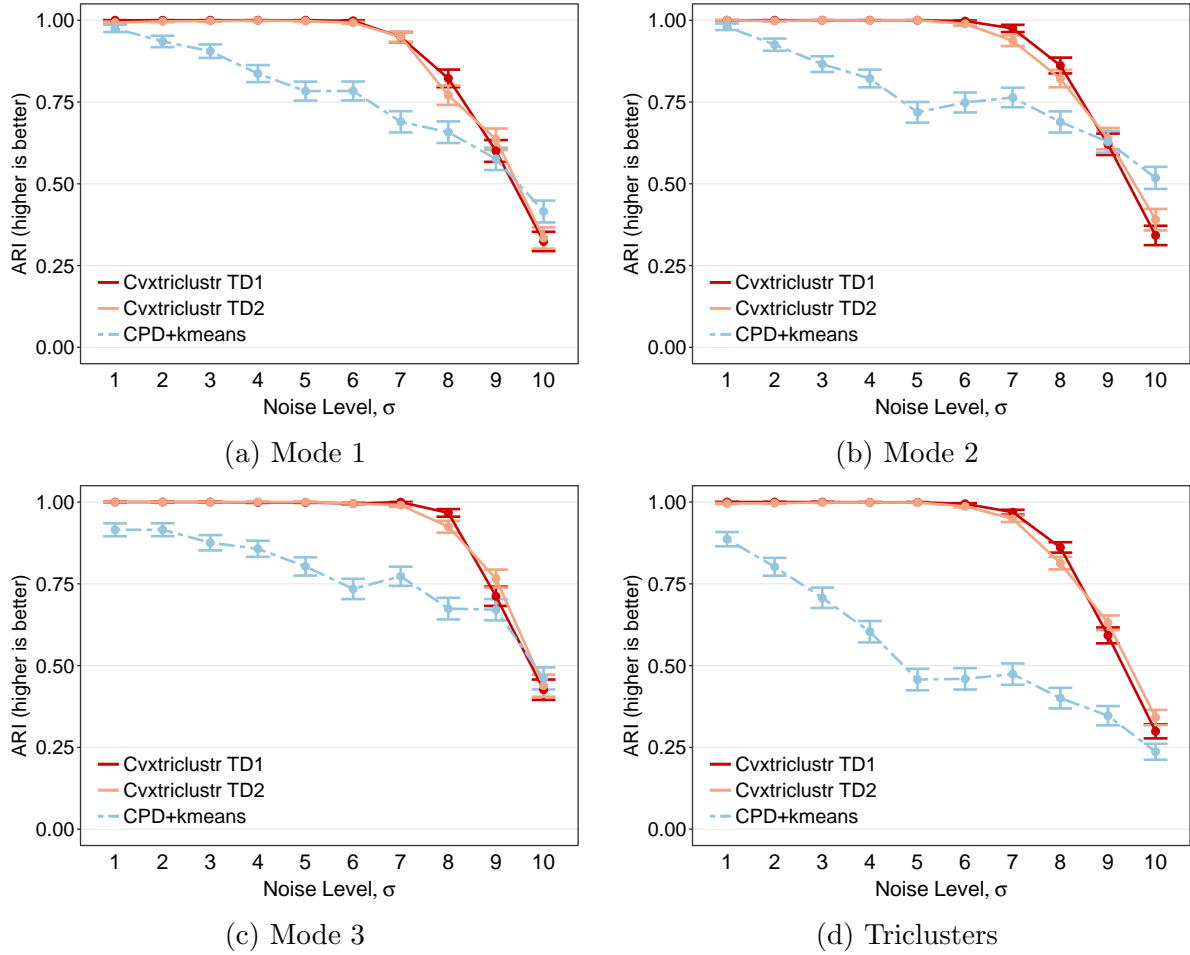


Figure B.3: Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error.

$I_1 = I_2 = I_3 = 60$, Variation of Information

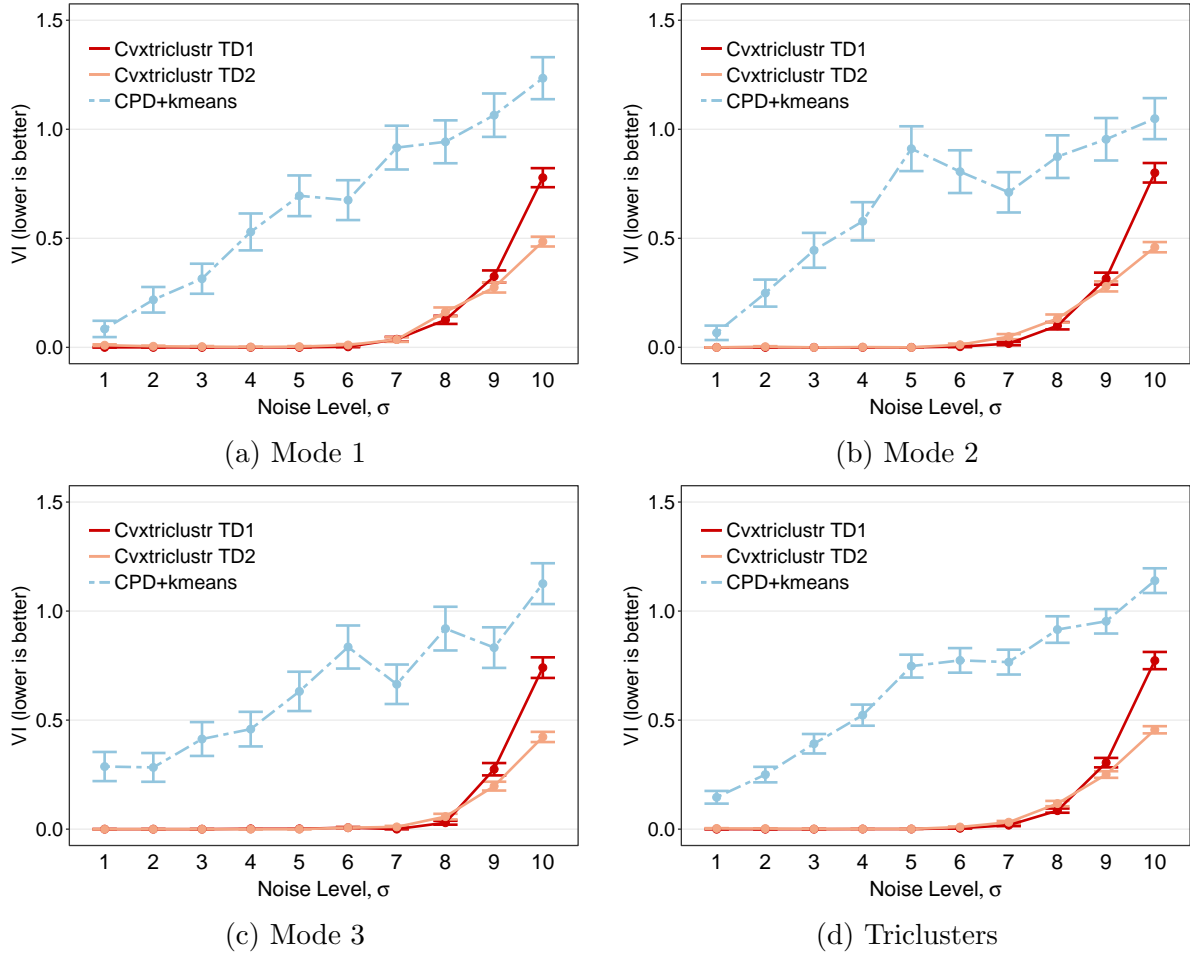


Figure B.4: Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error.

$I_1 = I_2 = I_3 = 80$, Adjusted Rand Index

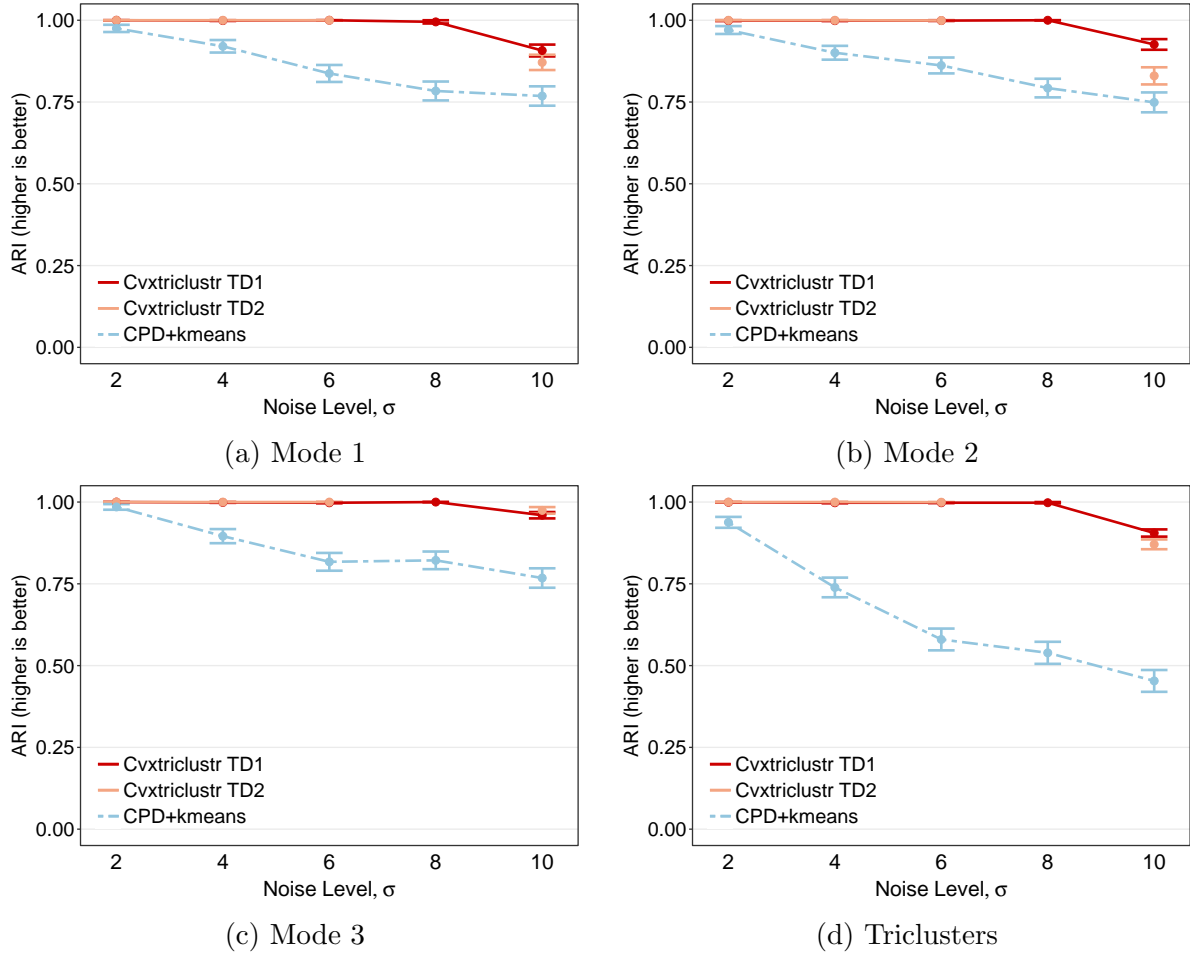


Figure B.5: Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 80$. Average adjusted rand index plus/minus one standard error.

$I_1 = I_2 = I_3 = 80$, Variation of Information

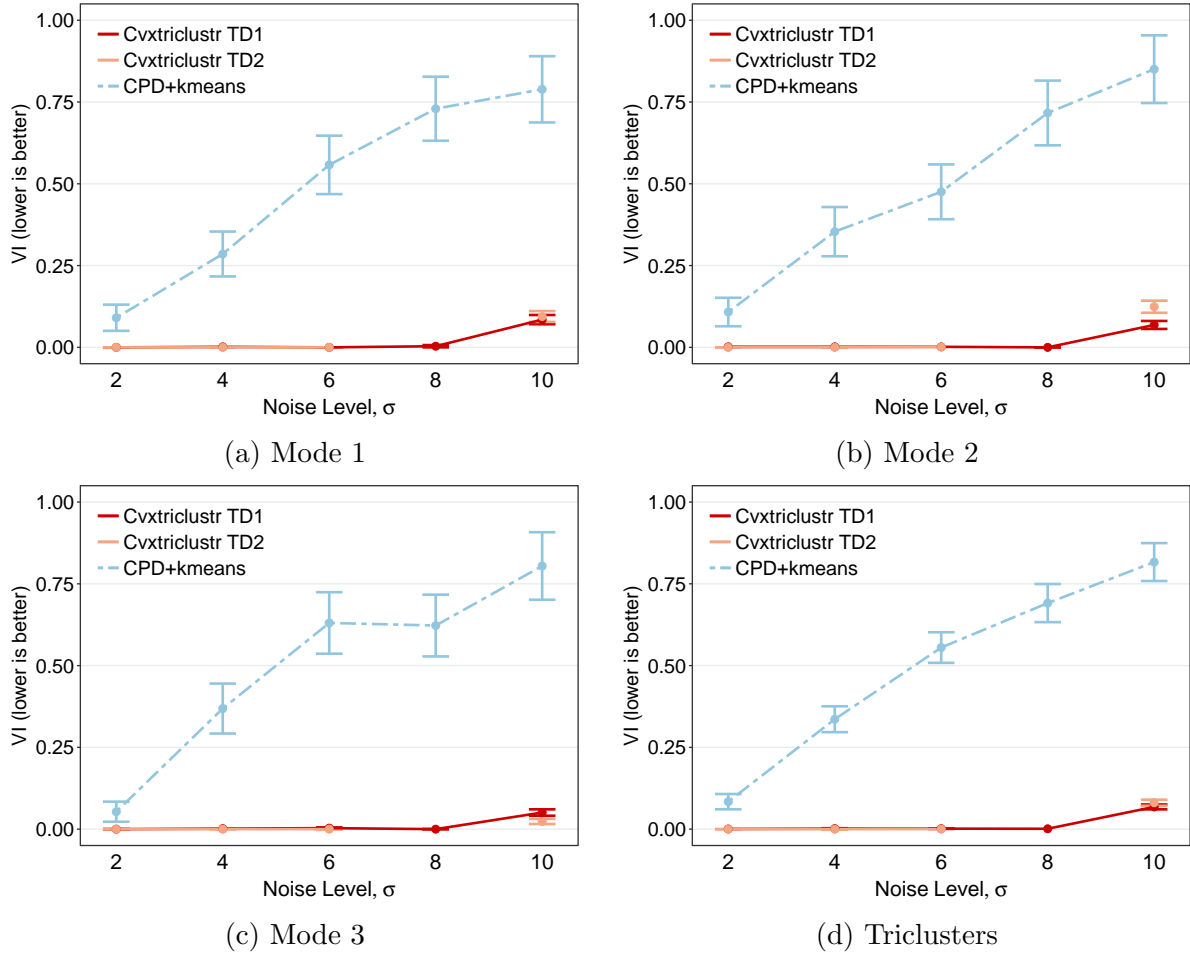


Figure B.6: Checkbox Simulation Results: Impact of Noise Level. Two balanced clusters per mode across different levels of homoskedastic noise for $I_1 = I_2 = I_3 = 80$. Average variation of information plus/minus one standard error.

B.2 Checkbox Pattern: Imbalanced Sizes and Homoskedasticity

Low Noise, Adjusted Rand Index

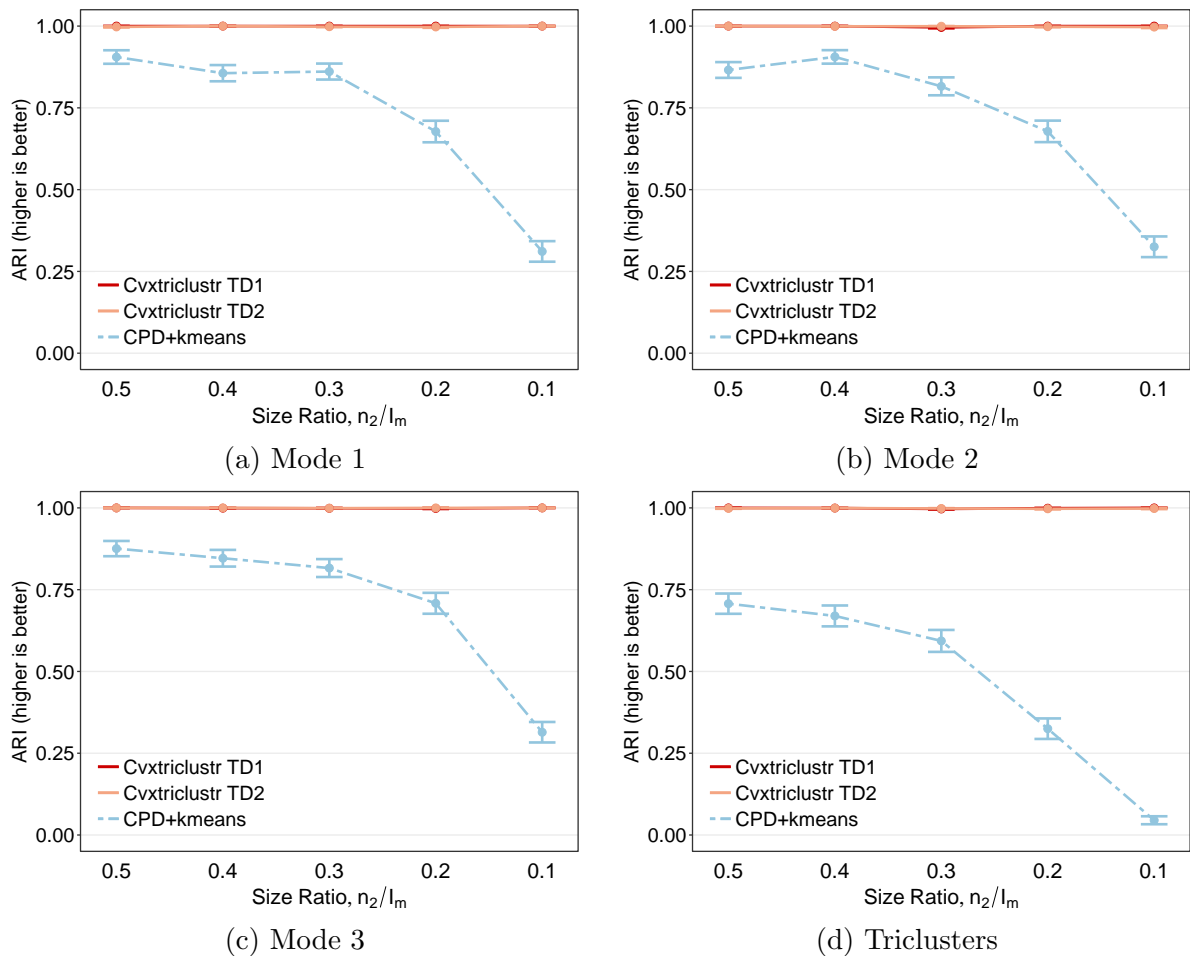


Figure B.7: Checkbox Simulation Results: Impact of Cluster Size Imbalance with Low Noise. Two imbalanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.

Low Noise, Variation of Information

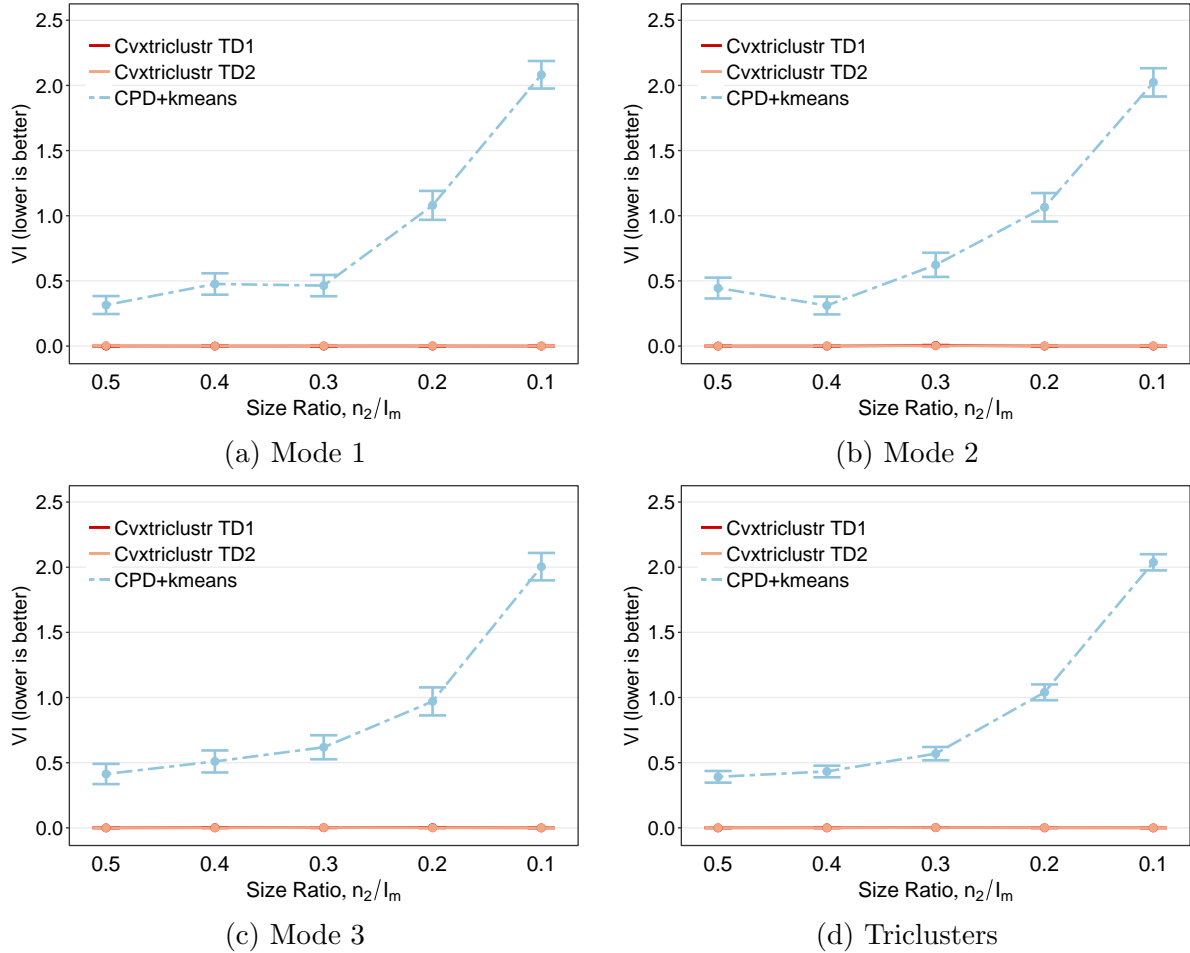


Figure B.8: Checkbox Simulation Results: Impact of Cluster Size Imbalance with Low Noise. Two imbalanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.

High Noise, Adjusted Rand Index

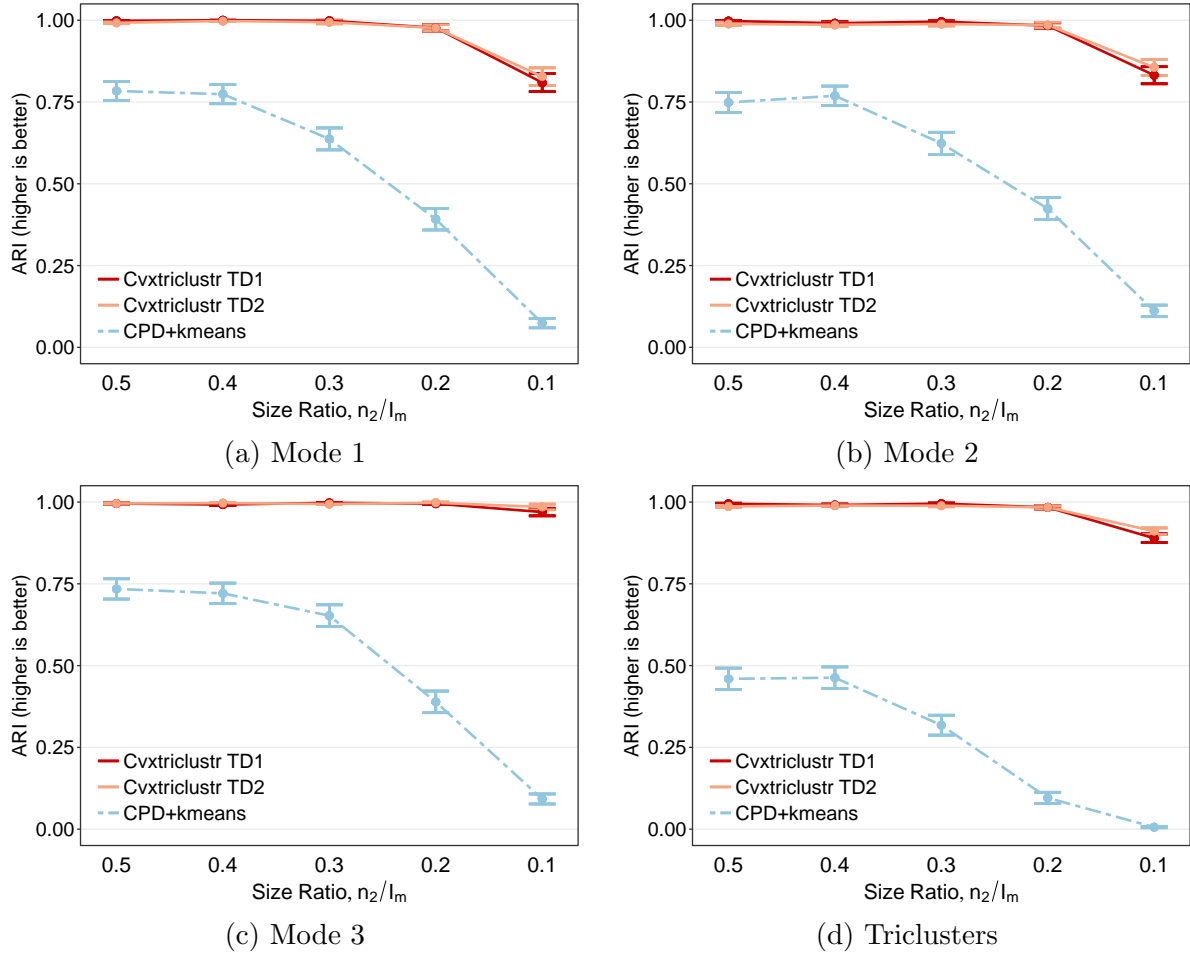


Figure B.9: Checkbox Simulation Results: Impact of Cluster Size Imbalance with High Noise. Two imbalanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.

High Noise, Variation of Information

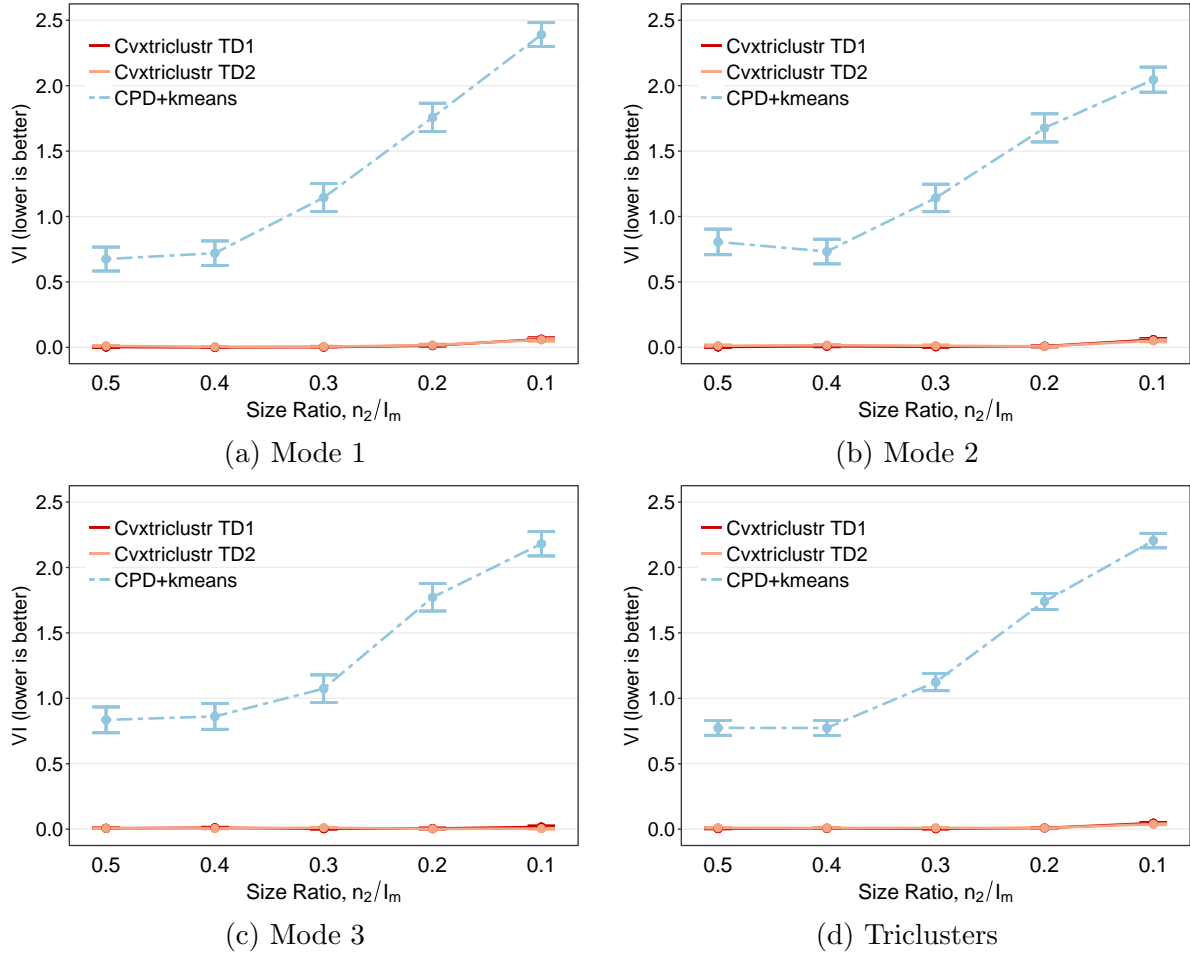


Figure B.10: Checkbox Simulation Results: Impact of Cluster Size Imbalance with High Noise. Two imbalanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) homoskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different degrees of cluster size imbalance. Size ratio = 0.5 corresponds to balanced clusters.

B.3 Checkbox Pattern: Balanced Sizes and Heteroskedasticity

Low Noise, Adjusted Rand Index

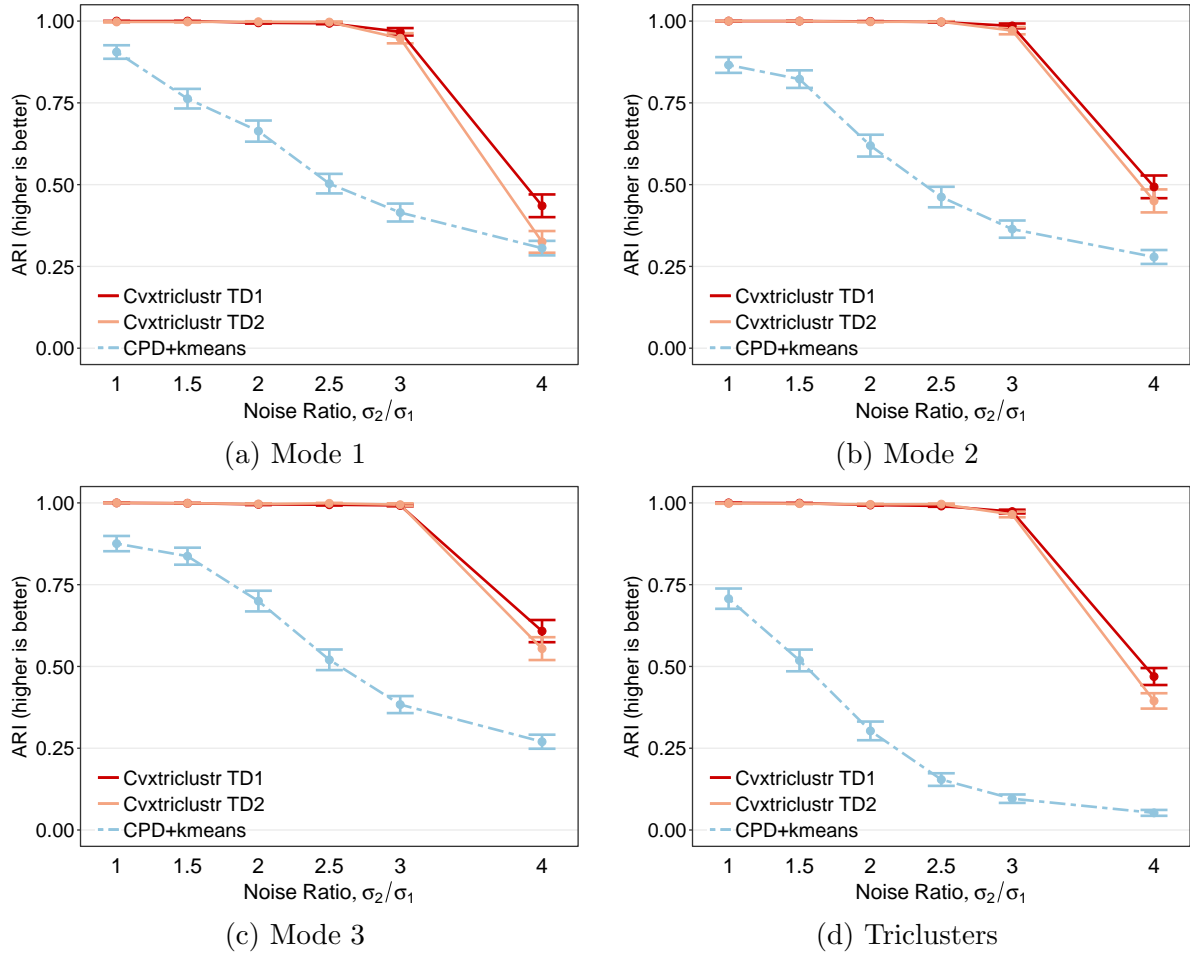


Figure B.11: Checkbox Simulation Results: Impact of Heteroskedasticity with Low Noise. Two balanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.

Low Noise, Variation of Information

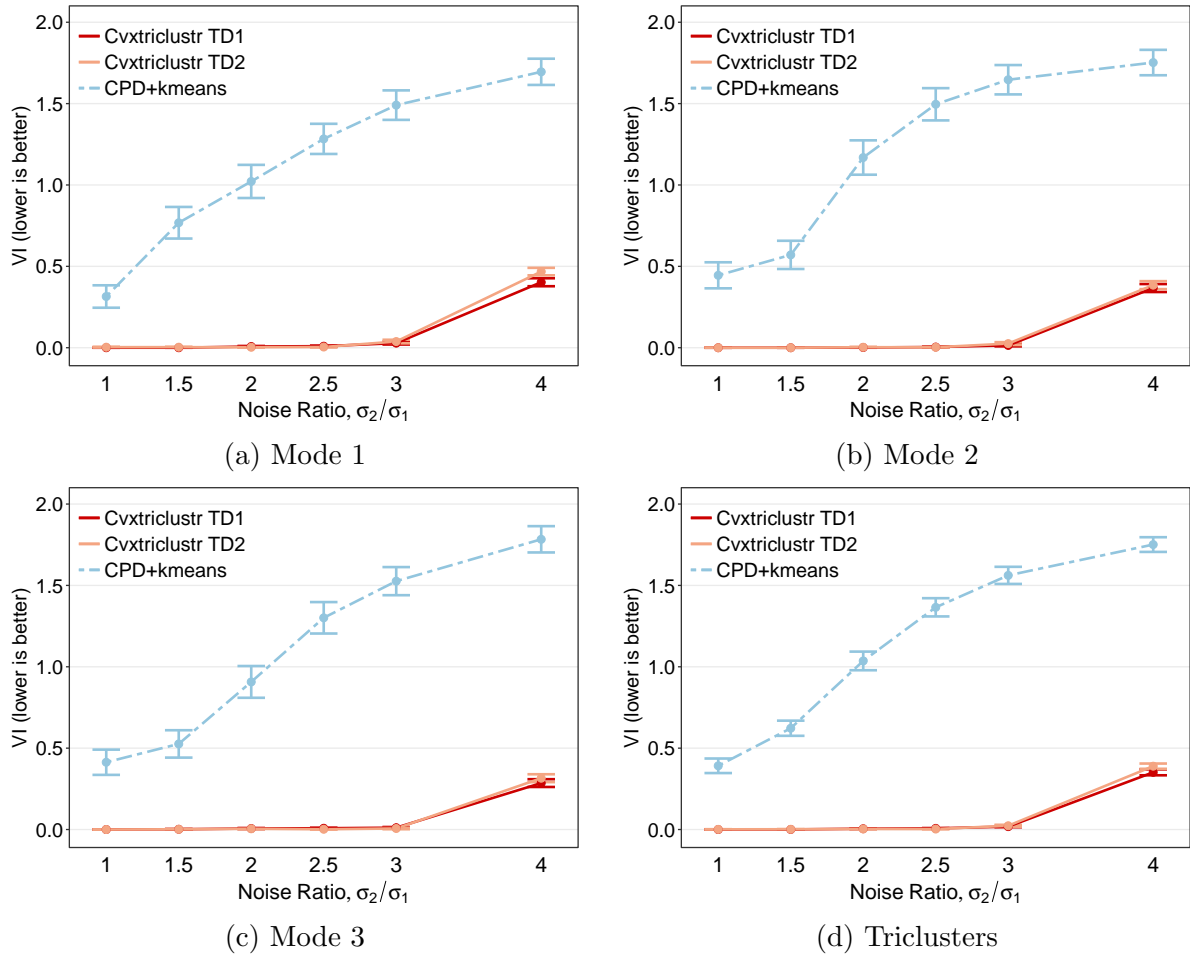


Figure B.12: Checkbox Simulation Results: Impact of Heteroskedasticity with Low Noise. Two balanced clusters per mode with low ($\sigma = 3$, $\text{SNR} = \frac{1}{3}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.

High Noise, Adjusted Rand Index

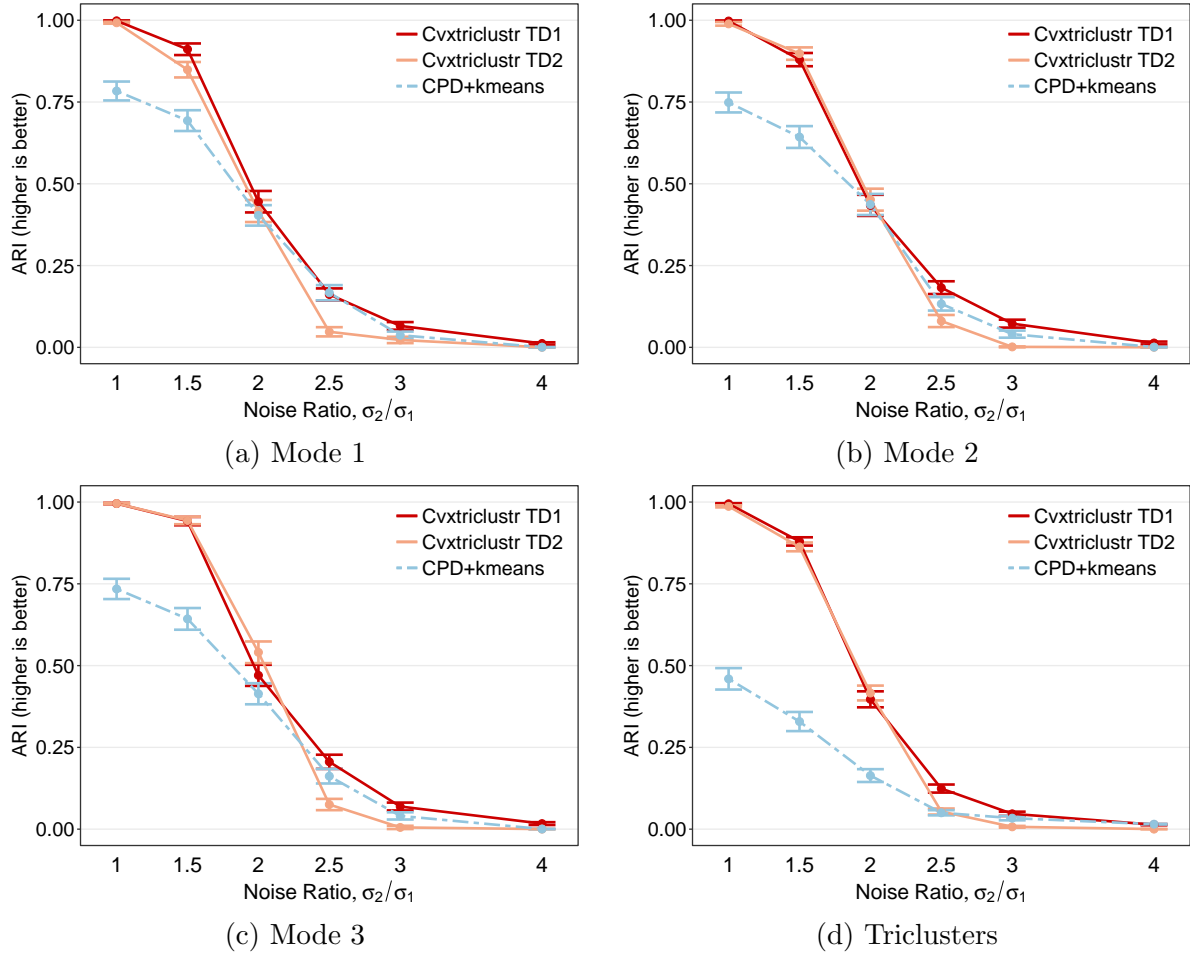


Figure B.13: Checkbox Simulation Results: Impact of Heteroskedasticity with High Noise. Two balanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average adjusted rand index plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.

High Noise, Variation of Information

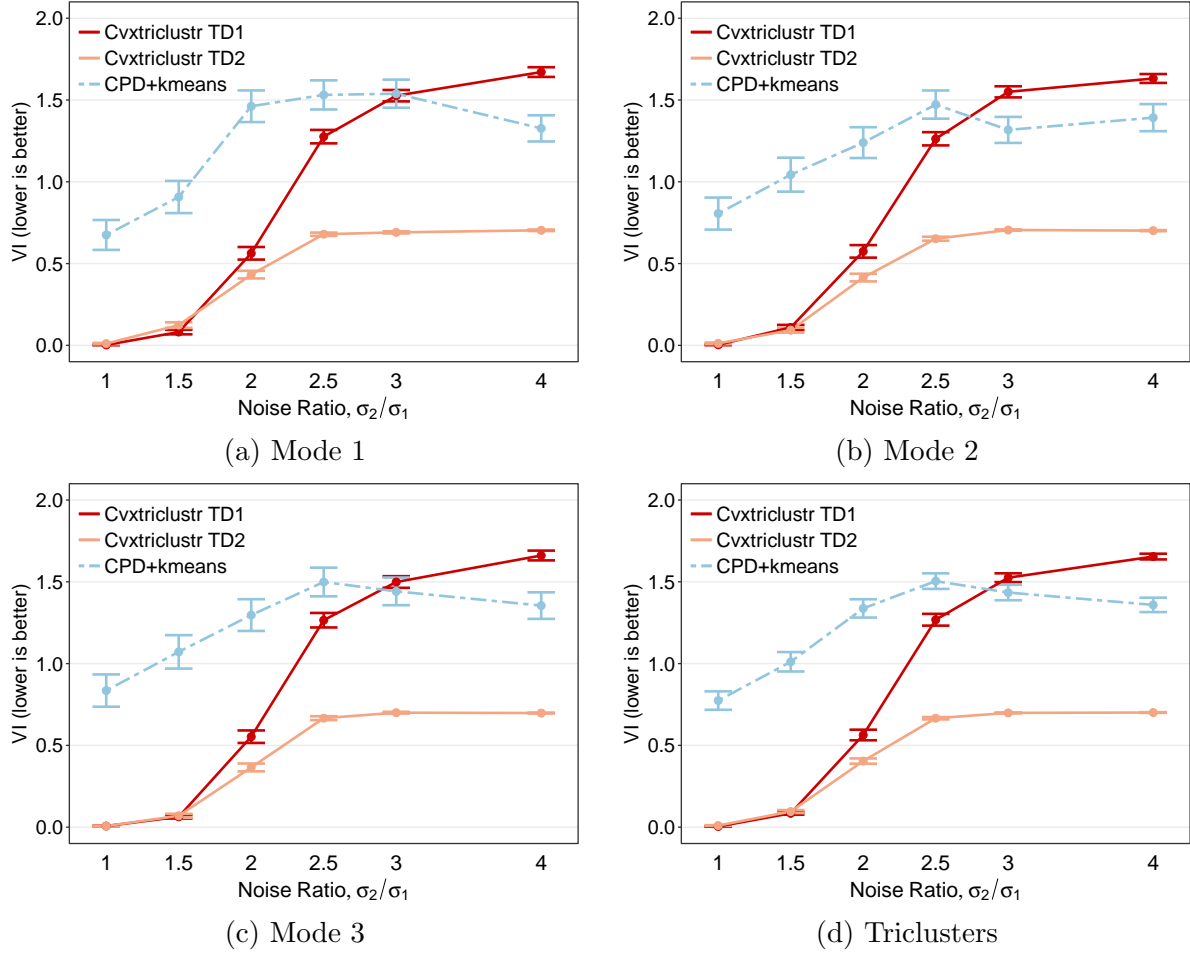


Figure B.14: Checkbox Simulation Results: Impact of Heteroskedasticity with High Noise. Two balanced clusters per mode with high ($\sigma = 6$, $\text{SNR} = \frac{1}{6}$) heteroskedastic noise and $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different levels of heteroskedasticity. Noise ratio = 1 corresponds to homoskedastic noise.

B.4 Different Clustering Structures

Adjusted Rand Index

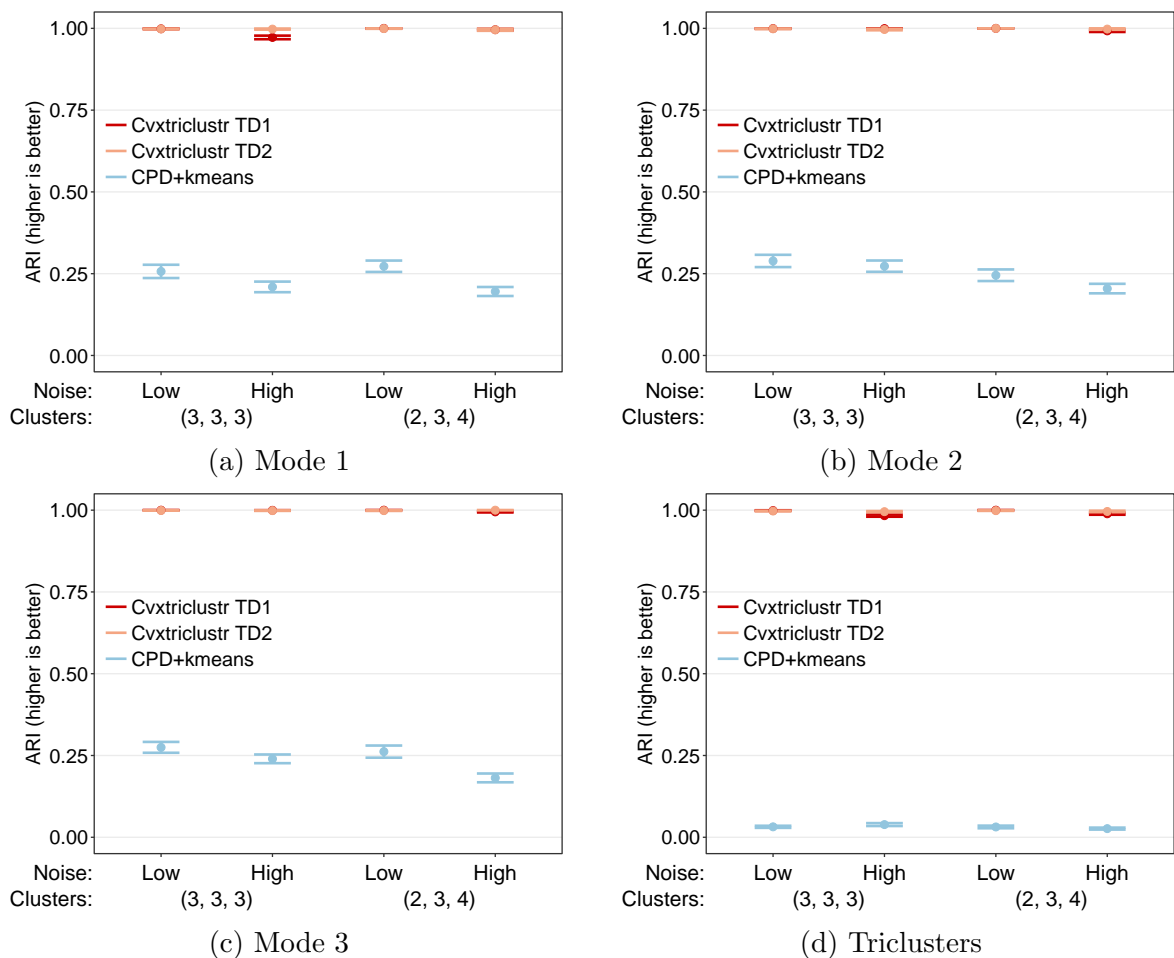


Figure B.15: Checkbox Simulation Results: Impact of Clustering Structure. Different number of balanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average triclustering performance plus/minus one standard error for different clustering structures, corresponding to either three clusters per mode or two, three, and four clusters along modes one, two, and three. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$).

Variation of Information

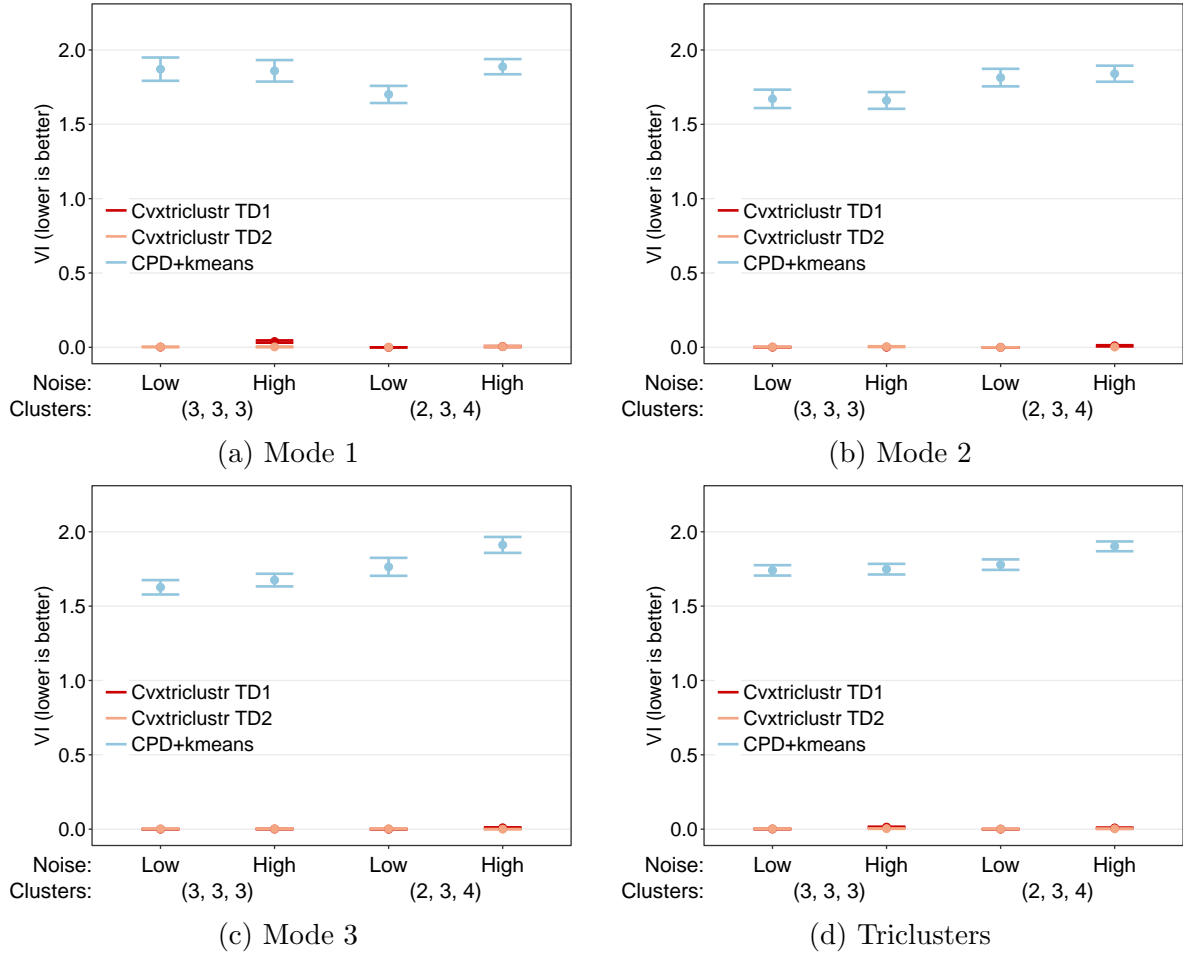


Figure B.16: Checkbox Simulation Results: Impact of Clustering Structure. Different number of balanced clusters per mode with either low or high homoskedastic noise for $I_1 = I_2 = I_3 = 60$. Average variation of information plus/minus one standard error for different clustering structures, corresponding to either three clusters per mode or two, three, and four clusters along modes one, two, and three. Low noise corresponds to $\sigma = 3$ ($\text{SNR} = \frac{1}{3}$) while high noise refers to $\sigma = 6$ ($\text{SNR} = \frac{1}{6}$).

B.5 Rectangular Tensors

Mode Lengths: Two Short and One Long, Adjusted Rand Index

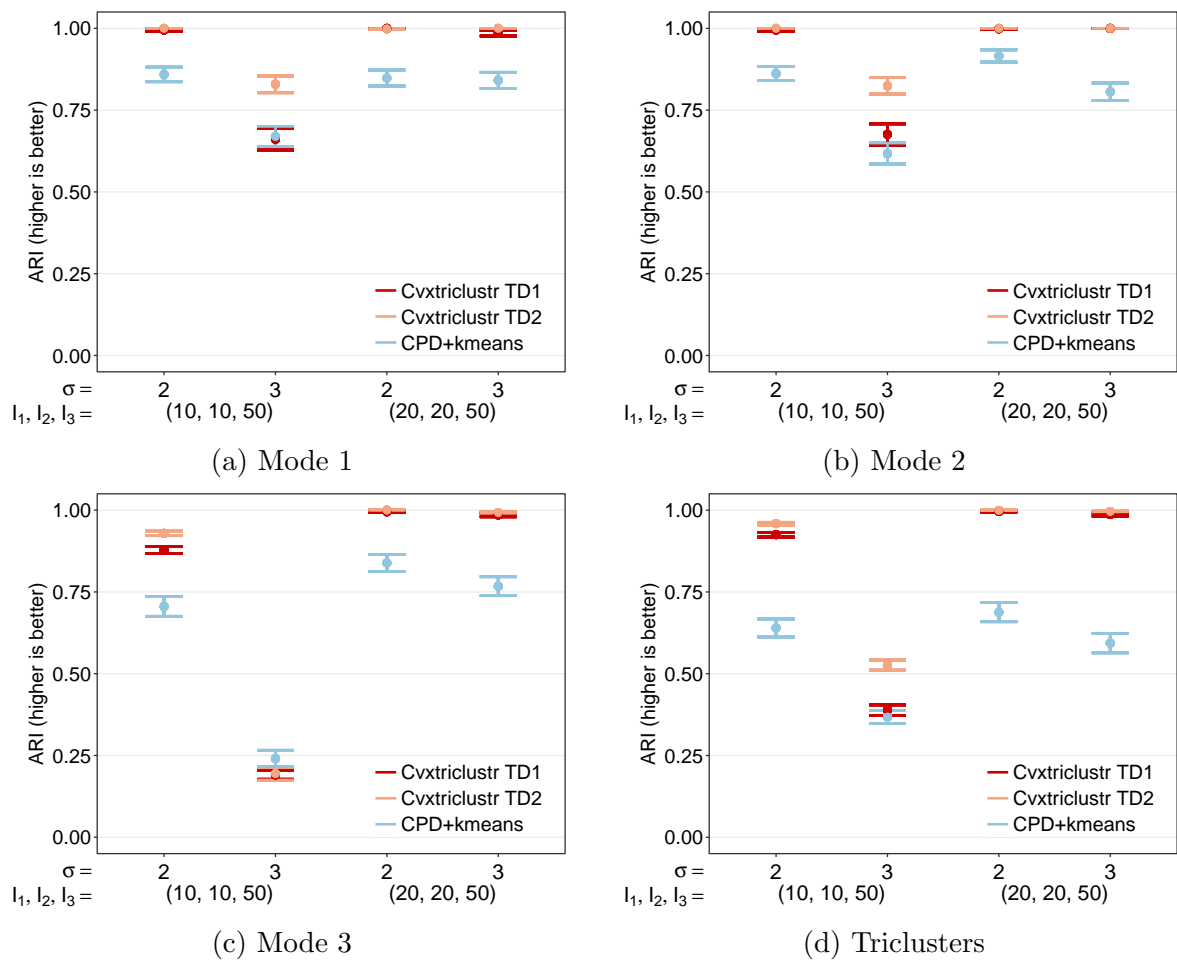


Figure B.17: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with two short modes and one longer mode. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.

Mode Lengths: Two Short and One Long, Variation of Information

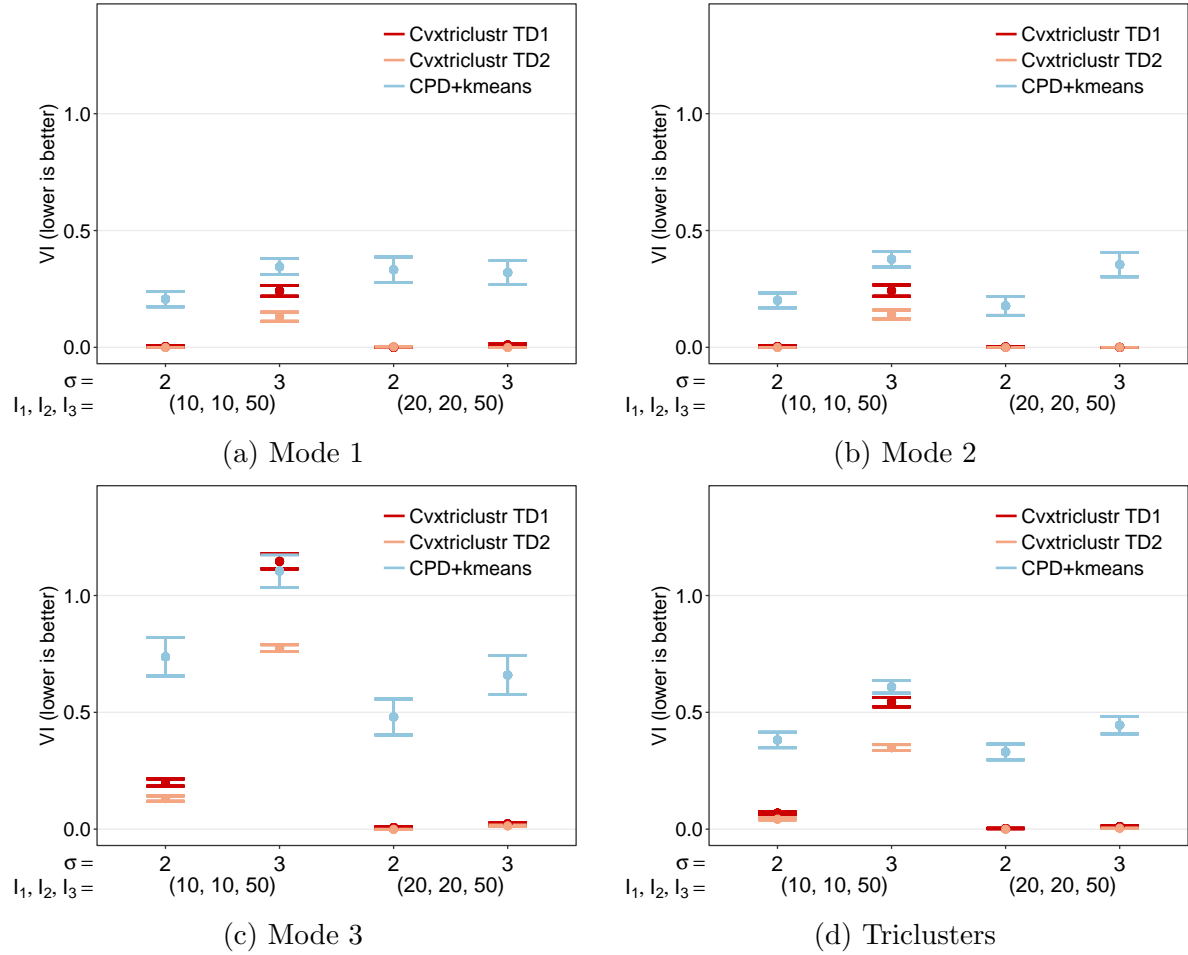


Figure B.18: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with two short modes and one longer mode. Average variation of information plus/minus one standard error for different noise levels and mode lengths.

Mode Lengths: One Short and Two Long, Adjusted Rand Index

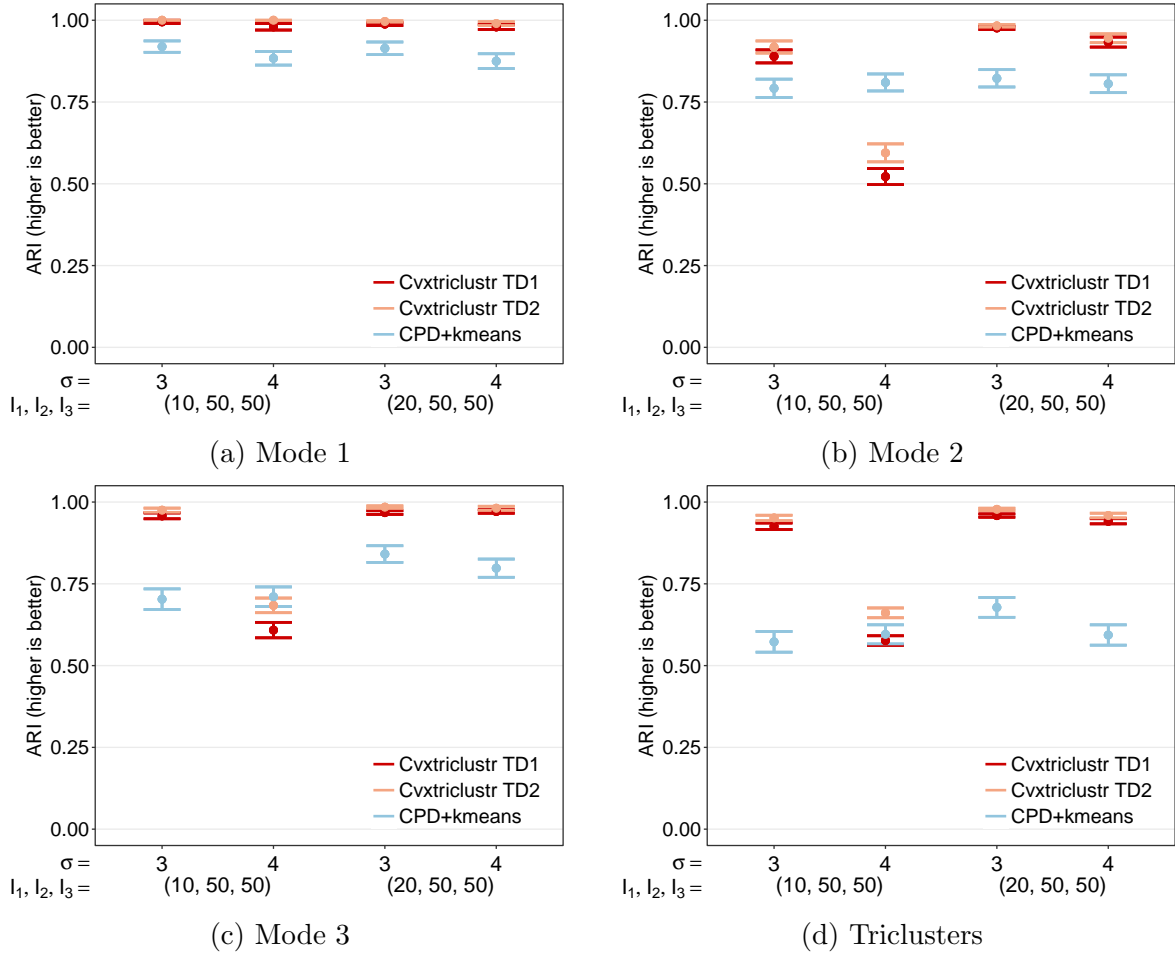


Figure B.19: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with one short mode and two longer modes. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.

Mode Lengths: One Short and Two Long, Variation of Information

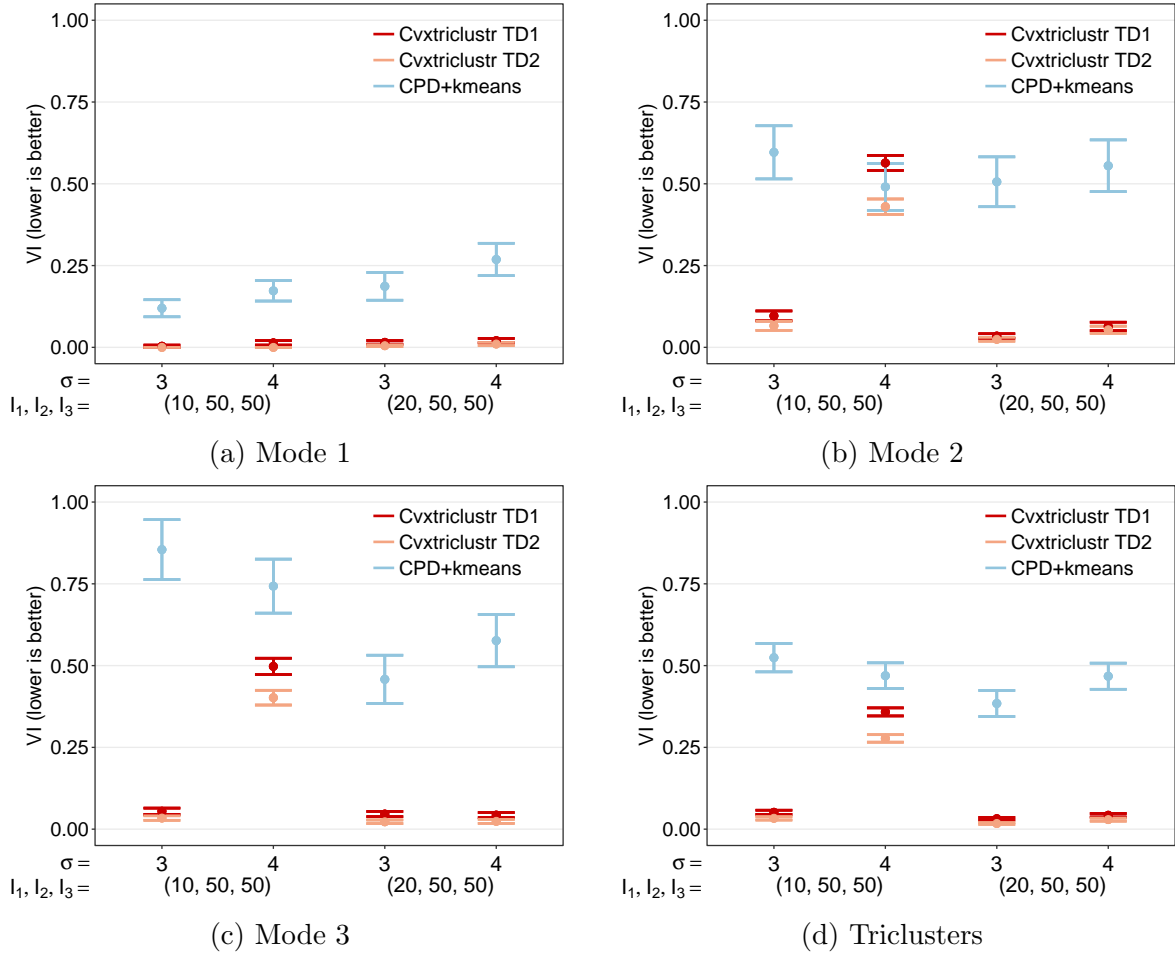


Figure B.20: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with one short mode and two longer modes. Average variation of information plus/minus one standard error for different noise levels and mode lengths.

Mode Lengths: Short, Medium, and Long, Adjusted Rand Index

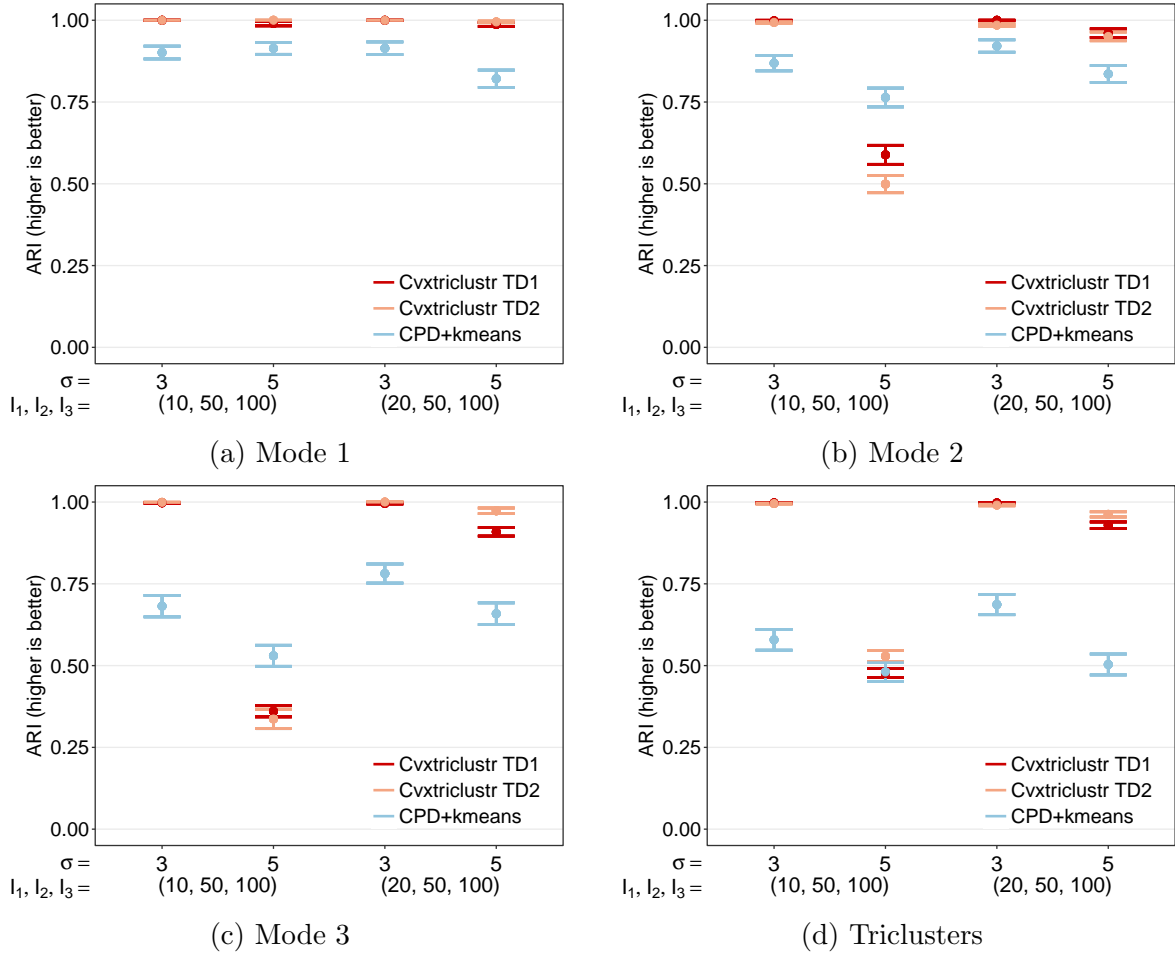


Figure B.21: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with short, medium, and long mode lengths. Average adjusted rand index plus/minus one standard error for different noise levels and mode lengths.

Mode Lengths: Short, Medium, and Long, Variation of Information

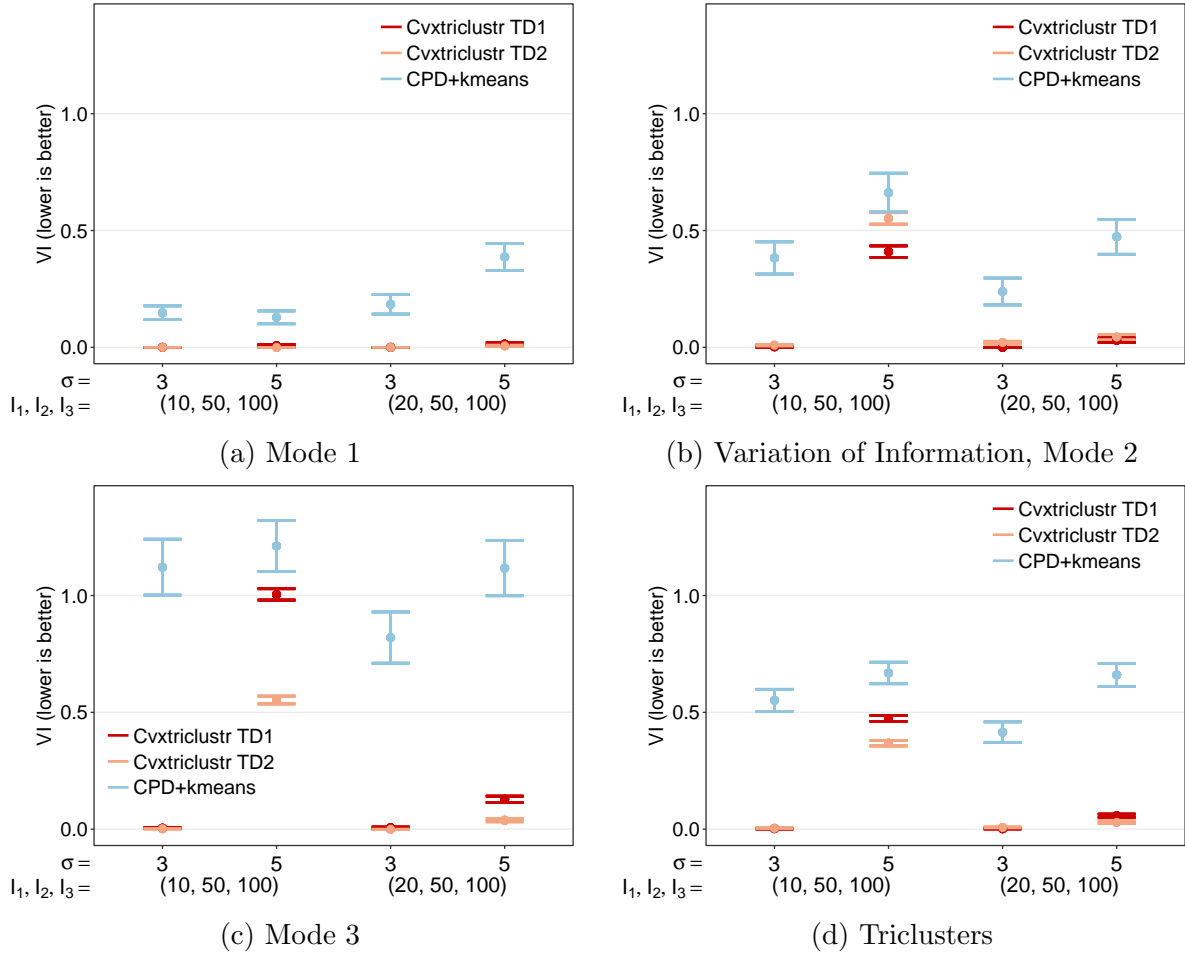


Figure B.22: Checkbox Simulation Results: Impact of Tensor Shape. Two balanced clusters per mode with two levels of homoskedastic noise for a tensor with short, medium, and long mode lengths. Average variation of information plus/minus one standard error for different noise levels and mode lengths.

B.6 CP Model, Adjusted Rand Index

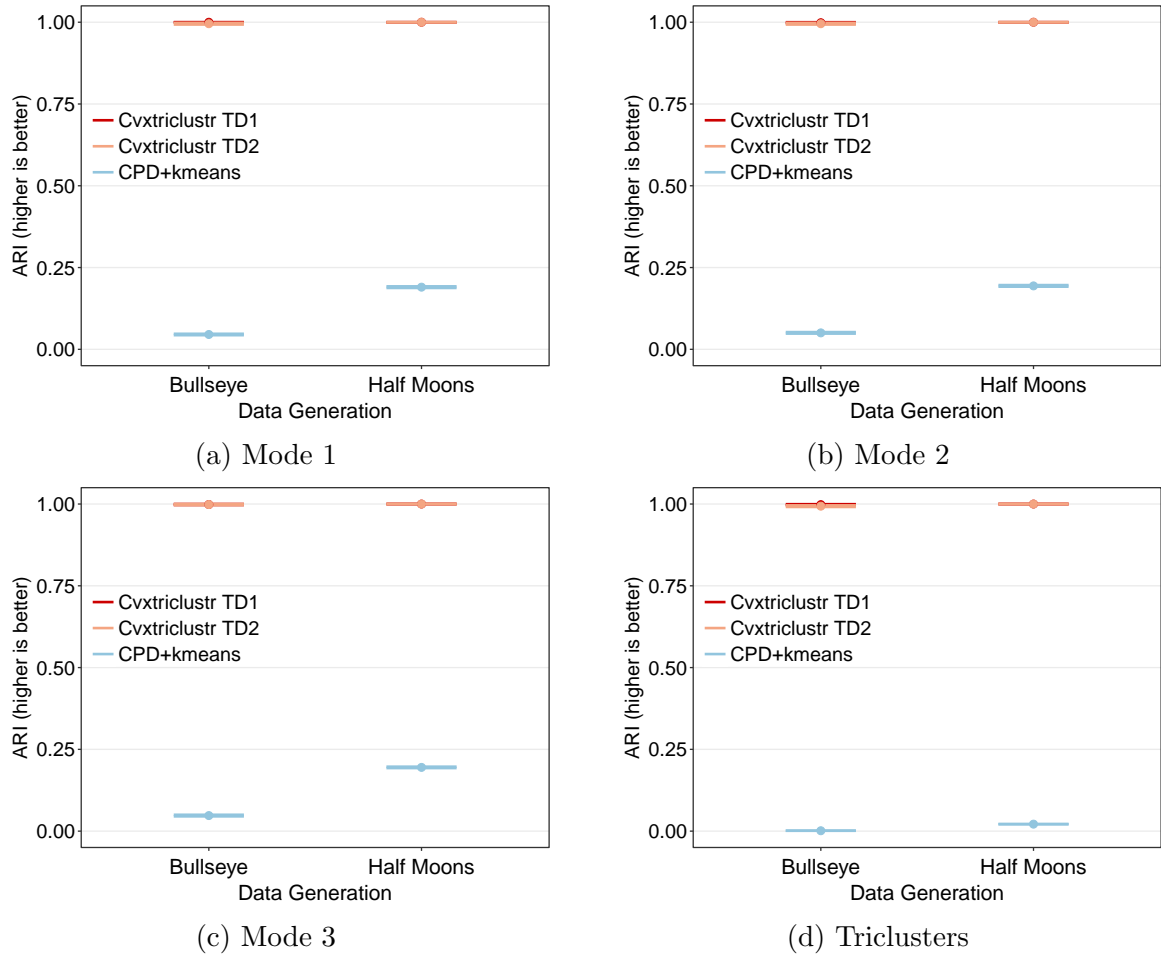


Figure B.23: CP Model Simulation Results. Two balanced clusters per mode with low homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average adjusted rand index plus/minus one standard error for two different data generation approaches. “Bullseye” and “Half Moons” refer to the shape embedded in the factor matrices used to generate the true tensor (Figure 4.15).

B.7 CP Model, Variation of Information

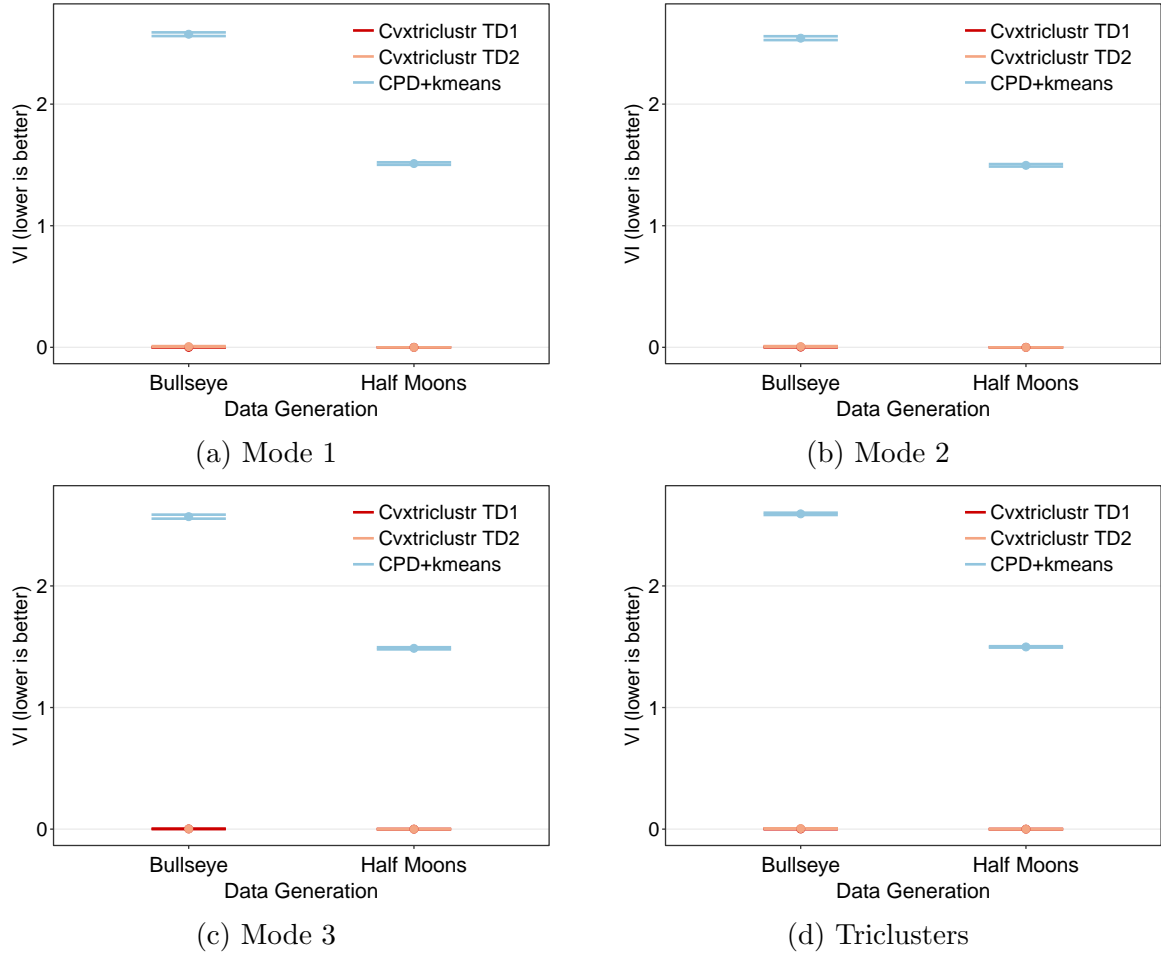


Figure B.24: CP Model Simulation Results. Two balanced clusters per mode with low homoskedastic noise for $I_1 = I_2 = I_3 = 40$. Average variation of information plus/minus one standard error for two different data generation approaches. “Bullseye” and “Half Moons” refer to the shape embedded in the factor matrices used to generate the true tensor (Figure 4.15).