

# UC San Diego

## UC San Diego Electronic Theses and Dissertations

### Title

Topological Based Machine Learning Methods

### Permalink

<https://escholarship.org/uc/item/4vr8963d>

### Author

Georges, Alexander

### Publication Date

2019

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA SAN DIEGO

**Topological Based Machine Learning Methods**

A dissertation submitted in partial satisfaction of the  
requirements for the degree  
Doctor of Philosophy

in

Physics

by

Alex Georges

Committee in charge:

Professor David Meyer, Chair  
Professor Benjamin Grinstein, Co-Chair  
Professor Melvin Leok  
Professor John McGreevy  
Professor Tom Murphy  
Professor Frank Würthwein

2019

Copyright  
Alex Georges, 2019  
All rights reserved.

The dissertation of Alex Georges is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

---

---

---

---

---

Co-Chair

---

Chair

University of California San Diego

2019

## DEDICATION

for my family, who is always there for me

for soupy, who was the best

for snoopy, who was the worst

and for everyone that doesn't take life too seriously

EPIGRAPH

*There is nothing either good or bad, but thinking makes it so.*

—Shakespeare

## TABLE OF CONTENTS

	Signature Page . . . . .	iii
	Dedication . . . . .	iv
	Epigraph . . . . .	v
	Table of Contents . . . . .	vi
	List of Figures . . . . .	ix
	List of Tables . . . . .	xiii
	Acknowledgements . . . . .	xiv
	Vita . . . . .	xv
	Abstract of the Dissertation . . . . .	xvi
Chapter 1	Introduction . . . . .	1
	1.1 Scope of the Dissertation . . . . .	2
Chapter 2	Data Science Fundamentals . . . . .	4
	2.1 Common Terms . . . . .	4
	2.2 What is Data? . . . . .	7
	2.2.1 Preparing Data . . . . .	9
	2.2.2 Curses . . . . .	10
	2.3 What is Data Science? . . . . .	11
	2.3.1 Typical Workflow . . . . .	14
	2.3.2 Bias-Variance Tradeoff . . . . .	16
	2.3.3 Ethics . . . . .	19
	2.4 Algorithm Classes and Techniques . . . . .	19
	2.4.1 Optimization . . . . .	20
	2.4.2 Dimensionality Reduction . . . . .	21
	2.4.3 Clustering . . . . .	23
	2.4.4 Classification . . . . .	25
	2.4.5 Topological Data Analysis . . . . .	26
Chapter 3	Dimensionality Reduction on Congress . . . . .	32
	3.1 Data . . . . .	33
	3.2 Summary of Results . . . . .	34
	3.3 Principal Component Analysis and Singular Value Decomposition . . . . .	35
	3.3.1 Information Flow over Time . . . . .	36

	3.3.2	Optimal Rank Flow over Time . . . . .	40
	3.4	Summary . . . . .	48
	3.5	Acknowledgements . . . . .	49
	3.6	Additional Figures for Information and Dimension Flows . . . . .	49
Chapter 4		Learned Persistent Homology . . . . .	56
	4.1	Background . . . . .	57
	4.1.1	Homology . . . . .	57
	4.1.2	Persistent Homology . . . . .	59
	4.2	Ambiguities in Persistence . . . . .	64
	4.3	Data . . . . .	68
	4.4	Learning Persistence . . . . .	70
	4.4.1	Supervised Learning of Persistence . . . . .	70
	4.4.2	Unsupervised Learning of Persistence . . . . .	72
	4.5	Ambiguities in Persistence: Part Deux! . . . . .	79
	4.6	Acknowledgements . . . . .	80
Chapter 5		Learned Mappers . . . . .	81
	5.1	Introduction . . . . .	81
	5.1.1	What is Mapper? . . . . .	83
	5.1.2	Mapper Algorithm . . . . .	84
	5.2	Training and Testing Procedures . . . . .	87
	5.2.1	Training Procedure . . . . .	89
	5.2.2	Mapping Unseen Points to the Committee . . . . .	90
	5.2.3	Splitting the Dataset . . . . .	92
	5.3	Numerical Experiments and Results . . . . .	93
	5.3.1	Main Results . . . . .	93
	5.3.2	Mapper Committee Dimensions . . . . .	95
	5.3.3	Description of Noise Models . . . . .	95
	5.3.4	Architecture of the End Classifier . . . . .	98
	5.3.5	Discussion . . . . .	100
	5.4	Theoretical Results . . . . .	101
	5.4.1	Proof of the MC Robustness . . . . .	102
	5.5	Conclusion . . . . .	103
Chapter 6		Conclusion and Future Directions . . . . .	105
Appendix A		Feature-Based Algorithm Selection for Mixed Integer Programming . . . . .	107
	A.1	Introduction . . . . .	108
	A.1.1	Mixed Integer Programming . . . . .	108
	A.1.2	Algorithm Selection . . . . .	109
	A.1.3	Related Work . . . . .	110
	A.2	Method . . . . .	111



A.2.1	Description of Data . . . . .	111
A.2.2	Upper Bounds/Well-Defined Features . . . . .	114
A.3	Feature Analysis . . . . .	115
A.3.1	Pearson Correlation Coefficients . . . . .	115
A.3.2	Principal Component Analysis . . . . .	116
A.3.3	Multidimensional Scaling . . . . .	118
A.3.4	Feature Investigation Conclusions . . . . .	120
A.4	Algorithm Selection Methods . . . . .	123
A.4.1	AdaBoost . . . . .	123
A.4.2	Hydra . . . . .	124
A.4.3	Selecting a Portfolio vs selecting an algorithm . . . . .	124
A.4.4	Performance Metric . . . . .	125
A.5	Results . . . . .	126
A.5.1	Performance . . . . .	126
A.5.2	Primal-Dual Integrals as a Proxy for Time . . . . .	128
A.5.3	Software package . . . . .	129
A.6	Conclusion . . . . .	130
A.7	Extracted Features . . . . .	131
A.7.1	Static features . . . . .	132
A.7.2	Dynamic features . . . . .	133
A.7.3	Top features for $M\&C$ . . . . .	134
A.7.4	Top features for $Regions$ . . . . .	135
A.8	Completion Codes . . . . .	135
Appendix B	Renormalization Group Flows and Morse Homology . . . . .	137
B.1	Introduction . . . . .	137
B.2	Background . . . . .	138
B.2.1	Classical Morse Theory . . . . .	139
B.2.2	Morse-Smale Homology . . . . .	140
B.2.3	Extensions to Morse Homology . . . . .	145
B.3	The $c$ -theorems . . . . .	146
B.3.1	The $d = 2$ $c$ -theorem . . . . .	146
B.3.2	The even dimensional $c$ -theorem . . . . .	147
B.3.3	The proposed even dimensional $c$ -theorem . . . . .	149
B.3.4	The $a$ -theorem . . . . .	152
B.4	Acknowledgements . . . . .	155
Appendix C	Explanations of Select Algorithms . . . . .	156
Bibliography	. . . . .	158

## LIST OF FIGURES

Figure 2.1:	In this diagram: deep learning is seen as a subset of machine learning, machine learning as intersecting AI, and data science as intersecting all the above. Relative sizes in this diagram have little meaning. If you want to pull in money, it's best to describe what you do as <i>AI</i> , since the term carries buzz.	12
Figure 2.2:	A “not-so-quite” accurate depiction of how computer science, math & statistics, and domain sciences interact.	14
Figure 2.3:	A simplified workflow of a data science project.	15
Figure 2.4:	The bias-variance tradeoff depicted by errors as a function of model complexity.	18
Figure 2.5:	Both of these datasets are topologically equivalent to a circle. TDA in general attempts to capture this information.	28
Figure 3.1:	The PCA retained information as a function of dimension given by Equation (3.1). The input data is the roll call vote for the 110th Senate (years 2007-2009).	38
Figure 3.2:	The PCA retained information as a function of dimension given by Equation (3.1). The input data is the roll call vote for the 110th House (years 2007-2009).	39
Figure 3.3:	The flow in $I(d = 1)(t)$ for the Senate for years 1789 to 2015 (1st to 113th Congress).	41
Figure 3.4:	The flow in $I(d = 1)(t)$ for the House for years 1789 to 2015 (1st to 113th Congress).	41
Figure 3.5:	The flow in $I(d = 1)(t)$ of the Senate roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1979 Congress. This figure is Figure 3.3 restricted to more recent years.	42
Figure 3.6:	The flow in $I(d = 1)(t)$ of the House roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1969 Congress. This figure is Figure 3.4 restricted to more recent years.	43
Figure 3.7:	Flow in the dimension of the Senate roll call data for years 1789 to 2015 (1st to 113th Congress).	45
Figure 3.8:	Flow in the dimension of the House roll call data for years 1789 to 2015 (1st to 113th Congress).	45
Figure 3.9:	Flow in the dimension of the Senate roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1979 Congress. This figure is Figure 3.7 restricted to more recent years.	46
Figure 3.10:	Flow in the dimension of the House roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1979 Congress. This figure is Figure 3.8 restricted to more recent years.	47
Figure 3.11:	The flow in $I(d = 1)(t)$ of the Senate roll call data for years 1789 to 1849 (1st to 30th Congress).	49
Figure 3.12:	The flow in $I(d = 1)(t)$ of the Senate roll call data for years 1847 to 1909 (30th to 60th Congress).	50

Figure 3.13:	The flow in $I(d = 1)(t)$ of the Senate roll call data for years 1907 to 1969 (60th to 90th Congress).	50
Figure 3.14:	Flow in the dimension of the Senate roll call data for years 1789 to 1849 (1st to 30th Congress).	51
Figure 3.15:	Flow in the dimension of the Senate roll call data for years 1847 to 1909 (30th to 60th Congress).	51
Figure 3.16:	Flow in the dimension of the Senate roll call data for years 1907 to 1969 (60th to 90th Congress).	52
Figure 3.17:	The flow in $I(d = 1)(t)$ of the House roll call data for years 1789 to 1849 (1st to 30th Congress).	52
Figure 3.18:	The flow in $I(d = 1)(t)$ of the House roll call data for years 1847 to 1909 (30th to 60th Congress).	53
Figure 3.19:	The flow in $I(d = 1)(t)$ of the House roll call data for years 1907 to 1969 (60th to 90th Congress).	53
Figure 3.20:	Flow in the dimension of the House roll call data for years 1789 to 1849 (1st to 30th Congress).	54
Figure 3.21:	Flow in the dimension of the House roll call data for years 1847 to 1909 (30th to 60th Congress).	54
Figure 3.22:	Flow in the dimension of the House roll call data for years 1907 to 1969 (60th to 90th Congress).	55
Figure 4.1:	Consider the 2-sphere as the topological space which can be seen as constructed from 2-simplices. The boundary operator acts on these simplices iteratively, at each step producing a set of lower dimensional simplices. Note, this figure leaves out the fact that $\partial^2 \equiv 0$ .	58
Figure 4.2:	A depiction of how the Rips complex changes as a function of $\epsilon$ . 0-simplices are represented by the individual data points, 1-simplices by the black edges connecting them, 2-simplices by yellow, 3-simplices by green, 4-simplices by red, and 5-simplices by blue. Image from [48].	60
Figure 4.3:	Three spaces that are topologically distinct and characterized by their set of Betti numbers. The red lines represent potential 1-dimensional generators. In the case of $S^2$ the generator can be contracted to a point, so $\beta_1 = 0$ . The generators for the tori cannot be contracted, hence $\beta_1 \neq 0$ for both.	64
Figure 4.4:	500 randomly sampled points from a circle.	65
Figure 4.5:	The barcode plot for Fig 4.4. For this example, $\delta_R \approx 11.6$ The top barcode represents how $H_0$ changes with respect to filtration value; the bottom barcode represents the change in $H_1$ . There is no significance to the ordering in either y-axis.	66
Figure 4.6:	A 2D Gaussian distribution used as a noise source. The Gaussian has $\sigma = 0.18$ for each axis.	66
Figure 4.7:	A circle + Gaussian noise. The circle diameter is 1 and the Gaussian has $\sigma = 0.18$ for each axis.	67

Figure 4.8:	The barcode plot for Fig 4.7. For this example, the two largest relative dominances are: $\delta_R \approx 0.23, 0.19$ . The top barcode represents how $H_0$ changes with respect to filtration value; the bottom barcode represents the change in $H_1$ . There is no significance to the ordering in either y-axis. . . . .	67
Figure 4.9:	A GMM with four different covariance matrices (see Appendix C) applied to classifying noisy circles and figure-8's with $\sigma = 0.1$ . Depicted here is a projection of the GMM and first two relative dominances to $\mathbb{R}^2$ . . . . .	71
Figure 4.10:	Test accuracy as a function of $\sigma$ for a GMM, RF, and NN. Even at $\sigma = 0.5$ , the RF and NN are able to predict the topology with 75% accuracy on test instances. The NN we use is a shallow multilayer perceptron ("MLP"). . .	72
Figure 4.11:	Noisy circles for two values of $\sigma$ . At $\sigma = 0.3$ accuracy $\sim 70\%$ and at $\sigma = 0.5$ accuracy $\sim 75\%$ for both the RF and NN. . . . .	73
Figure 4.12:	Noisy figure-8's for two values of $\sigma$ . At $\sigma = 0.3$ accuracy $\sim 70\%$ and at $\sigma = 0.5$ accuracy $\sim 75\%$ for both the RF and NN. . . . .	73
Figure 4.13:	$p$ -value with respect to noise for pairwise comparisons of $\delta_R^{(i)}$ for the circle. . . . .	75
Figure 4.14:	$p$ -value with respect to noise for pairwise comparisons of $\delta_R^{(i)}$ for the figure-8. . . . .	76
Figure 4.15:	Histograms for the 3 largest relative dominances of the circle. At each noise level, blue is $\delta_R^{(1)}$ , green is $\delta_R^{(2)}$ , red is $\delta_R^{(3)}$ . These colors do not correspond to the $p$ -value measurements. . . . .	77
Figure 4.16:	Histograms for the 3 largest relative dominances of the figure 8. At each noise level, blue is $\delta_R^{(1)}$ , green is $\delta_R^{(2)}$ , red is $\delta_R^{(3)}$ . These colors do not correspond to the $p$ -value measurements. . . . .	78
Figure 5.1:	Illustration of our MC method. In practice we use not just one Mapper object but a whole committee of mappers (see Section 5.2). . . . .	83
Figure 5.2:	Mapper objects computed for 10k MNIST training data, with contractive autoencoder projection (see Section 5.2). Color corresponds to projection value, and node size corresponds to total number of points in the node. $n_{\text{int}} = n_{\text{bins}} = 10$ for the LHS figure, $n_{\text{int}} = n_{\text{bins}} = 20$ for the RHS figure. . .	85
Figure 5.3:	Histograms of length scales at which clusters are grouped via single-linkage clustering. These plots represent varied $n_{\text{bins}}$ for the first interval in the open cover of $U_\alpha$ , using the filter $f = \text{PCA}_1$ . In this case, increasing $n_{\text{bins}}$ from 10 to 20 has no effect on the cutoff value, which is set to 6.6. . . . .	86
Figure 5.4:	Latent space representations of two nodes in the compressed layer of the VAE (i.e., a 2-dimensional subspace of the projection to 20-dimensions). We use a $\beta$ -term that multiplies the KL divergence. $\beta = 0$ yields the best overall robustness. . . . .	88
Figure 5.5:	Test accuracy of MC trained using 30k sample from MNIST dataset. We present the accuracy with respect to the number of PCA filters used in the committee of mappers, and the number of subsets the whole 30k sample was split into. . . . .	93
Figure 5.6:	The normalized accuracy with respect to $l^2$ -norm. The PCA based Mapper approach far outperforms the CNN approach for all the noise models. . . .	96

Figure 5.7:	The normalized accuracy with respect to $l^2$ -norm. The PCA based Mapper approach far outperforms the CNN approach for all the noise models. For 60k MNIST, we choose to investigate just two Mapper based methods: PCA and VAE which seem to perform the best on average. . . . .	97
Figure 5.8:	The number 7 as a function of $\lambda$ for the Gauss blur noise model. $\lambda$ can be thought of as a percentage of 28, a fundamental length scale in this data. . .	98
Figure 5.9:	The number 7 as a function of $\lambda$ for the s&p noise model. $\lambda = 0.01$ , for instance, corresponds to a 1% chance of flipping a pixel. . . . .	99
Figure 5.10:	The number 7 as a function of $\lambda$ for the Gaussian noise model. . . . .	99
Figure A.1:	A Pearson correlation coefficient heatmap computed using the first 20 dimensions in the feature space. The diagonal and upper triangular elements have been removed since this heatmap is a symmetric plot. . . . .	117
Figure A.2:	The PCA retained information as a function of dimension given by Equation (3.1). The input data is the static+dynamic feature space. The various horizontal lines are 70%, 90% and 95% thresholds. The 95% threshold occurs at $d = 71$ . . . . .	118
Figure A.3:	PCA components 1 and 2, for just the dynamic features for M&C. Notice the distinct 2 clusters that emerge. . . . .	119
Figure A.4:	PCA components 1 and 2, for static and dynamic features for M&C. Notice, the cluster information is washed away. . . . .	120
Figure A.5:	The MDS retained information as a function of dimension given by equation (A.2). The input data is the static+dynamic feature space. The 95% threshold occurs at $d = 19$ . . . . .	121
Figure A.6:	MDS projection to $d = 2$ , for just the dynamic features for M&C. Notice the distinct 2 clusters that emerge. . . . .	122
Figure A.7:	MDS projection to $d = 2$ , for static and dynamic features for M&C. Notice, the cluster information is washed away. . . . .	122
Figure A.8:	Portfolio performance on the M&C data set. Selectors were trained and tested on Primal-Dual integral values. . . . .	127
Figure A.9:	Portfolio performance on the M&C data set. Selectors were trained on Primal-Dual integral values and tested on their time performance. . . . .	128
Figure B.1:	Torus with level sets and critical points drawn in. With the appropriate coordinate system, the upmost points correspond to the largest heights. . .	139
Figure B.2:	Torus with level sets and a few examples of the stable and unstable manifolds. The topmost ring depicts just one possible flow coming from the $\lambda = 2$ point, but there is indeed a two dimensional structure flowing from this point. . .	142

## LIST OF TABLES

Table 4.1:	Parameters of Generated Data . . . . .	69
Table 4.2:	The $p$ -values we would like to see for the circle and figure-8 for the three comparisons in $\delta_R^{(i)}$ . A $p$ -value $\leq 5\%$ means the distributions are different; a $p$ -value $> 5\%$ means the distributions can be considered the same. . . . .	74
Table 4.3:	The data $\mathbb{X}$ is either the letter “A” or “O”. Theoretically, persistent homology will return a 1 for the first two Betti numbers; mapper will return a graph that looks different between the datasets. . . . .	80
Table 5.1:	The initial classification accuracies. PCA, CAE, DAE, VAE are all Mapper based approaches, and only differ in the type of projection. These values give the normalization in Equation (5.4). . . . .	95
Table 5.2:	The total number of nodes in the mapper committee, with respect to choice in latent space projections. . . . .	97
Table B.1:	The leading order dependence of $\chi$ and $W$ on the couplings for various physical theories. We specify the loop-order to which each calculation was done in [142]. . . . .	154

## ACKNOWLEDGEMENTS

My time at UCSD has been invaluable, not only for my academic and professional growth, but also for my personal growth. I have numerous people to thank that have invested their time into me, which has consequently helped lead to my success so far, for which I am very grateful. There are far too many people to name individually, and to those people I leave out I apologize.

First off, I would like to thank my family: Richard, Marlene, Marissa, Lauren, and all the friends I've made in San Diego for without whom, I would have gone crazy. Second, my advisors: David, Ben, and Frank, for without whom, I would still be just 1% as intelligent as them compared to the 2% that I am now. Third, the staff, including Sharmila, Catherine, Jasmyn, Kevin, and Lester, for without whom, all of us would be tripping over our shoelaces. Finally, I'd like to thank Jacek as we accomplished something truly amazing by drawing circles and lines.

Chapter 5, in part is currently being prepared for submission for publication of the material. Jacek Cyranka, Alex Georges, and David A. Meyer. The dissertation author was a primary investigator and author of this material.

Appendix A is in part is a reprint of the material as it appears in ZIB-Report 18-17 [1]. The dissertation author was a primary investigator and author of this paper. Alex Georges, Ambros Gleixner, Gorana Gojić, Robert Lion Gottwald, David Haley, Gregor Hendel, Bartłomiej Matejczyk. The work for this article was partly conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM) within the program “Graduate-Level Research in Industrial Projects for Students” (GRIPS) 2017. The described research activities have been partly funded by the Federal Ministry for Economic Affairs and Energy within the project BEAM-ME (ID: 03ET4023A-F). The authors would like to thank the Zuse Institute Berlin, the Institute for Pure and Applied Mathematics and the National Science Foundation for their contributions in resources and finances to this project. Special thanks to Daniel Hulme and Karsten Lehmann from Satalia for many fruitful discussions about the topic.

## VITA

- 2012 B. A. in Physics, University of California Berkeley
- 2012 B. A. in Pure Mathematics, University of California Berkeley
- 2019 Ph. D. in Physics, University of California San Diego

## PUBLICATIONS AND OTHER MATERIAL

Cyranka J., **Georges A.**, Meyer D. A. *Mapper Based Classifier*. Submitted for publication, 2019.

**Georges A.**, Gleixner A., Gojić G., Gottwald R., Haley D., Hendel G., Matejczyk B. *Feature-based algorithm selection for mixed integer programming*. Technical Report 18-17, ZIB, Takustr. 7, 14195 Berlin, 2018.

The BaBar collaboration, Lees, J.P., Poireau, V., Tisserand, V., Tico, J.G., Grauges, E., Palano, A., Eigen, G., Stugu, B., Brown, D.N., **Georges, A.** and Kerth, L.T., 2013. *Search for a low-mass scalar Higgs boson decaying to a tau pair in single-photon decays of  $\Upsilon(1S)$* . Physical Review D, 88(7), p.071102.

**Georges A.**, Kolomensky Y., Paudel U. *Search for the Tau-Pair Decays of a Light Scalar Higgs Boson in Radiative Transition of  $\Upsilon(1S) \rightarrow \gamma A^0$* . BaBar Analysis Document #2272.



ABSTRACT OF THE DISSERTATION

**Topological Based Machine Learning Methods**

by

Alex Georges

Doctor of Philosophy in Physics

University of California San Diego, 2019

Professor David Meyer, Chair  
Professor Benjamin Grinstein, Co-Chair

This dissertation presents novel approaches and applications of machine learning architectures. In particular, these approaches are based on tools from topological data analysis and are used in conjunction with conventional machine learning methods. Topological data analysis, which is based on algebraic topology, can identify significant global mathematical structures which are out of reach of many other approaches. When we use topology we benefit from generality, and when we use conventional methods we benefit from specificity.

This dissertation contains a broad overview of data science and topological data analysis, then transitions to three distinct machine learning applications of these methods. The first

application uses linear methods to discover the inherent dimensionality of the manifold given by congressional roll call votes. The second uses persistent homology to identify extremely noisy images in both supervised and unsupervised tasks. The last application uses mapper objects to produce robust classification algorithms. Two additional projects are presented later in the appendix, and are related to the three main applications. The first of these constructs a method to choose optimal optimizers, and the second places mathematical constraints on the structure of renormalization group flows.

# Chapter 1

## Introduction

Data science is such a rapidly changing field that by the time this sentence finishes, it will likely be outdated. A few key factors drive its acceleration, which include: the decreasing cost of hardware pertaining to data needs (i.e., data collection, processing, and storage) [2], the increase in scientific methods to analyze the data, and the repeated success shown by these methods in widespread domains. Andrew Ng, a prominent researcher in data science, has referred to artificial intelligence as “the new electricity.” Many state-of-the-art applications in data science are based on methods which are inherently geometric: that is, changing the parameter space or data can significantly change the overall outcome. There are methods, however, that are resistant to these changes and many of them fall within topological data analysis. Indeed, for any data science project at hand, there is no universal best algorithm to approach the problem - so described by the “No Free Lunch Theorem” [3]. Rather than choosing geometric *or* topological methods, this dissertation focuses on the combination of both, particularly as they pertain to machine learning. The terms *data science*, *machine learning*, and *artificial intelligence* will sometimes be used interchangeably, especially since the distinction between them may be nonexistent or unimportant depending on the project. However, in Section 2.3 we will see in what context these fields are different.

Data science is becoming more ubiquitous in almost all aspects of life and its transformative potential has been proven time and time again in widespread domains. Hopefully the work presented in this dissertation will, to some extent, push the boundary of how we interact with data science.

## **1.1 Scope of the Dissertation**

Chapter 2 is a broad overview of data science and the fundamental tools that will be used in later chapters, and should be considered a general introduction to these topics rather than an in-depth treatment. Chapter 3 is an applied analysis using a linear dimensionality reduction technique referred to as principal component analysis. The data analyzed consists of voting behavior of United States politicians in Congress, and the task is to uncover the true dimensionality of the data with respect to year. Chapter 3 is meant to be an introduction to principal component analysis, which will be used in Chapter 5 and Appendix A. Chapter 4 is the first analysis in the dissertation that utilizes methods from topological data analysis (“TDA”) in conjunction with conventional machine learning methods. The goal in this project is to classify data structures in extremely noisy data through both supervised and unsupervised approaches. By the end of Chapter 4, we will see some of the benefits and issues with this approach, and hence will move on to a more powerful technique in Chapter 5.

Chapter 5 should be considered the seminal work of this dissertation. It incorporates traditional methods from data science along with tools from topological data analysis, specifically mapper objects. The novel combination of methods presented here results in a robust algorithm which resolves some fundamental issues present in other current state-of-the-art methods. We apply the algorithm we develop to the task of noisy image classification.

Additional projects presented in this dissertation are contained in the appendix, and are related to the methods in previous chapters. Appendix A is a project in teaching a computer how

to select the optimal optimizing function to solve a given task. In Appendix B, we present a conjecture pertaining to renormalization group flows which shows an equivalence between the medium and strong versions of the  $a$ -theorem. This conjecture allows us to loosen a restrictive requirement that allows these flows to be written in terms of pure gradients. The mathematical tools we use in Chapter 5 and Appendix B both, in part, come from Morse theory.

# Chapter 2

## Data Science Fundamentals

This chapter is meant to be a short overview of the main concepts and tools from data science that will be used in following chapters. The approach to this discussion will be more intuitive, with specific technical details held off until later chapters. For comprehensive overviews on data science, machine learning, deep learning, and topological data analysis please see [4, 5, 6, 7], respectively. For a diagram depicting the differences between these fields, please see Figure 2.1.

This dissertation broadly focuses on using topological data analysis (“TDA”) in conjunction with other machine learning methods to produce robust algorithms, so it is necessary to understand these other algorithms as well. The reader that is familiar with general concepts in data science can skip the majority of this chapter, but may want to glance at Section 2.4.5 which introduces the concepts from TDA that will be used in future chapters. As a quick reminder of terms, please see Section 2.1.

### 2.1 Common Terms

This section is meant to be a reminder of some terms that are commonly used in data science. The limited glossary below is likely more useful for the reader that has some background

in this field, and would like a reminder of how certain concepts are defined.

- Bagging: A combination of implementing both *bootstrapping* and *ensemble averaging*. The word comes from the combination of bootstrapping+aggregating.
- Bootstrapping: Random sampling of *instances* from a data set with replacement. See *bagging, feature bagging*.
- Class: The category of *label* assigned to an *instance*.
- Classification: The process of learning a map  $f : X \rightarrow Y$ , from features to discrete labels. See *regression*.
- Cross Validation: “CV” is a method to construct an accurate representation of the testing error. In CV, we split the data in such a way so that our model sees more testing data. See *testing error, testing set, training error, training set*.
- Discriminative Learning: Attempts to learn a mapping from features to labels. Intuitively analogous to learning a distribution  $p(y|x)$ . Geometrically equivalent to learning boundaries between classes. See *generative*.
- Ensemble Averaging: Taking a collection of individual machine learning models, and building one final output from them.
- Feature: A piece of information describing some aspect of a data point. Considering the data as an  $m \times n$  matrix, the  $n$  features consist of the  $n$  columns.
- Feature Bagging: Random sampling of *features* from a data set with replacement. See *bagging, bootstrapping*.
- Generative Learning: Attempts to infer the underlying distribution that created the data. Intuitively analogous to learning a joint distribution  $p(x, y)$ . Geometrically equivalent to learning distributions of classes. See *discriminative*.

- **Hyperparameter:** A value not learned by the model and doesn't appear in the *loss* function. It can still have a significant impact on the overall performance of our algorithm. See *parameter*.
- **Instance:** The mathematical notation to denote an instance is typically  $x$ . See *label*.
- **Label:** In *supervised* learning, labels are continuous or discrete values that characterize the *instance* in some way. In the discrete case, *classes* distinguish the instances. The mathematical notation to denote a label is typically  $y$ .
- **Loss:** A function that associates some cost to a model's decisions. Optimization seeks to minimize this loss in order to reach our goal.
- **Overfitting:** When our overall procedure learns about individual data points too well. Generalization can suffer in this case. See *underfitting*.
- **Parameter:** A value learned by the model in order to minimize the *loss*. See *hyperparameter*.
- **Regression:** The process of learning a map  $f : X \rightarrow Y$ , from features to continuous labels. See *classification*.
- **Semi-supervised Learning:** Use both labeled and unlabeled data for training. Typically, training is done with less labeled than unlabeled.
- **Supervised Learning:** Use labels to inform the machine what the objects are that it's considering. Intuitively analogous to learning a distribution  $p(y|x)$ , given the *features*  $x$  and *labels*  $y$ . See *unsupervised* learning.
- **Testing Error:** The error associated with testing data, i.e., data the algorithm has not yet seen. This error gives a measure of how well the model generalizes. See *training error*.
- **Testing Set:** The data used to determine generalization performance of our algorithm. See *cross validation*, *training set*.



- Training Error: The error associated with the training data. A well-trained model can have exceedingly low training error. See *testing error*.
- Training Set: The data used to teach our algorithm. See *cross validation, testing set*.
- Underfitting: When our overall procedure doesn't learn enough about individual data points. Generalization can suffer in this case. See *overfitting*.
- Unsupervised Learning: No labels are used to inform the machine what the objects are that it's considering. Intuitively analogous to learning a prior distribution  $p(x)$  given the *feature* data only. See *supervised learning*.

## 2.2 What is Data?

Data can be anything. It can range from business analytics, to interpersonal relationships, to exploding stars. Regardless of the data we may be interested in, it remains to be seen whether we can do something useful with it. There are numerous factors that go into this, but we can broadly characterize these factors into: the quality of our data and the quality of our algorithm. In this section we will discuss data and in Section 2.3 we will discuss algorithms.

To have high quality data we must ensure that the information we are grabbing in the first place is relevant to us, and in the second place that this information can be mathematically represented in a way that is conducive to producing a high quality algorithm. And to have a high quality algorithm, we must choose an architecture that works well with our data and accomplishes a well defined goal.

For instance, let's say we are trying to design an algorithm that produces optimal seating arrangements at a wedding. The first piece of information we will need for this is, of course, who will be present at the wedding. We can already define a placing solely on this information by random placements, but this probably doesn't achieve our goal of optimality, whatever that may

be. If on the other hand the data we have is the table cloth colors, that probably won't help us out at all. In the former case, the information is relevant but not sufficient; in the second case, the information is irrelevant.

So let's add interpersonal relationships here, of which there's any number of ways to do. For now, let's say each pair of people is assigned +1 if they are friendly with one another, -1 if they are not, and 0 otherwise. We can mathematically represent this information in a few ways, for instance by an  $m \times m$  matrix where  $m$  represents the number of wedding guests, or by encoding this information in an  $m$ -dimensional vector for each person. In the former case, we would be dealing with an object in  $\mathbb{R}^{m \times m}$  and in the latter case we would have  $m$  objects in  $\mathbb{R}^m$ .

Using either mathematical encoding, we can analyze this new data and come up with an answer that's closer to optimality than just knowing people's names, but there's still the issue that complete strangers may be seated at the same table. From here, again we can do a number of things to get closer to the optimal seating arrangement. For the sake of argument let's consider two ways forward: (1) modify *just* the data to also include whether two individuals know each other and leave the fundamental optimality algorithm unchanged (2) add a penalty term in our optimality algorithm that prefers acquaintances to be seated at the same table. The first of these methods relies on modifying the data, the second more so on the algorithm. *A priori* there is likely no way of knowing which approach is best at this step or even at some of the others. For instance, perhaps this is not the best way to mathematically represent acquaintances in the first place because there is redundant information - if a pair of individuals has an interpersonal relationship of +1 or -1 then they obviously know each other, so we don't gain any information by inserting that they are acquaintances.

So at this point, we have skirted around a few fundamental questions: what information do we need to include in the data, how do we mathematically represent this information, how do we achieve the goal at hand, and why is the goal we have chosen the goal we should be considering? In the following sections, we will touch more on these last two questions. In most cases though,

the best answers to all these questions come holistically when we think about how the methods we use to analyze the data interact with the data itself. For instance, if I were to ask “how much data do we need to answer this question?” or “do we have too little or too much information describing the data?” the answer in all likelihood is dependent on the method with which we use to answer these questions. In this contrived wedding example, answering these questions might be exceedingly simple given the overall setup of the problem. In other cases though, the data or methods can be exceedingly complex and, in the first place, we might not even know whether the goal we have chosen is the one we should be using. In these cases, it is worth thinking more about these fundamental questions to hopefully lay down a holistic path to solving the problem. In practice, finding this path is an iterative procedure which can be seen in Figure 2.3.

Under the lens that most everything can be seen as data, and that something useful can be done with that data, it is no wonder that data and data science in general is becoming increasingly ubiquitous in all facets of life.

### **2.2.1 Preparing Data**

As seen in Section 2.2, one of the first and most important steps is to prepare the data we’ll use to achieve our goal. A non-exhaustive list of the steps that should be taken to prepare data include: defining what information is necessary, how to get this information, and how to solve issues with the data. These issues can range from missing data, corrupted data, or even just data that we don’t really know how to handle properly. It can be the case that data preparation is the most difficult and time consuming step, especially for industrial or applied projects. In this dissertation, we will be more concerned with the methods to analyze the data rather than the data preparation.

## 2.2.2 Curses

At first thought, it might seem that adding more and more features to data is the best thing to do, especially given the previous wedding example. This is true to an extent, given that the information we attach to a data point is relevant, but there are diminishing returns to this at some point. One of the concrete ways in which adding features ad infinitum can negatively impact the quality of a dataset is that it can become “cursed.” The *curse of dimensionality* refers to the phenomenon that exponentially more data is required to fill higher dimensional spaces.

Let’s say our data lives in a 1-dimensional space, and that we discretize this space so that the data is restricted to live on  $n$  lattice points. In this case, we will need  $n$  data points to fill the space. Now, let’s say we extend this discretized interval to 2-dimensions so that there are now  $n^2$  lattice sites. Again, we will need  $n^2$  points to fill this space. In general for this scenario, we will need  $n^d$  points to entirely fill the space.

Intuitively, if there is significant mathematical structure in the original dimension, we might assume the data fills a significant portion of the space. In other words, the value  $n^d$  gives the approximate number of data points we should have for  $d$ -dimensional data in order to find mathematical structure - note that this is indeed an upper bound as we assume the entire space is filled. Luckily, interesting datasets are not completely random and have some correlations in them which reduce the overall dimensionality. In images, for instance, we have a cohesive picture because individual pixel values are correlated to their neighboring pixels. The information contained in these correlations effectively restrict the originally  $d$ -dimensional data to subspaces of lower dimensions because the information living in each dimension is not independent of the other dimensions. In other words, if the correlations are strong, we should be able to predict neighboring pixel values, and hence not all pixel values are necessary to represent the image. See Section 2.4.2 for a further discussion on dimensionality reduction.

A different but related way in which more dimensions can reduce the data quality is that too many dimensions can lead to an overfitted model (see Section 2.3.2). When there are too

many features, our model may be able to learn about and discriminate between every single data point, which means the model is overfit.

Often times, we start out with high dimensional “cursed” data, project it to a lower dimensional space and work within that space (see Section 2.4.2). Additionally, by performing exploratory data analysis (Section 2.3.1) we can see the impact of the high dimensionality.

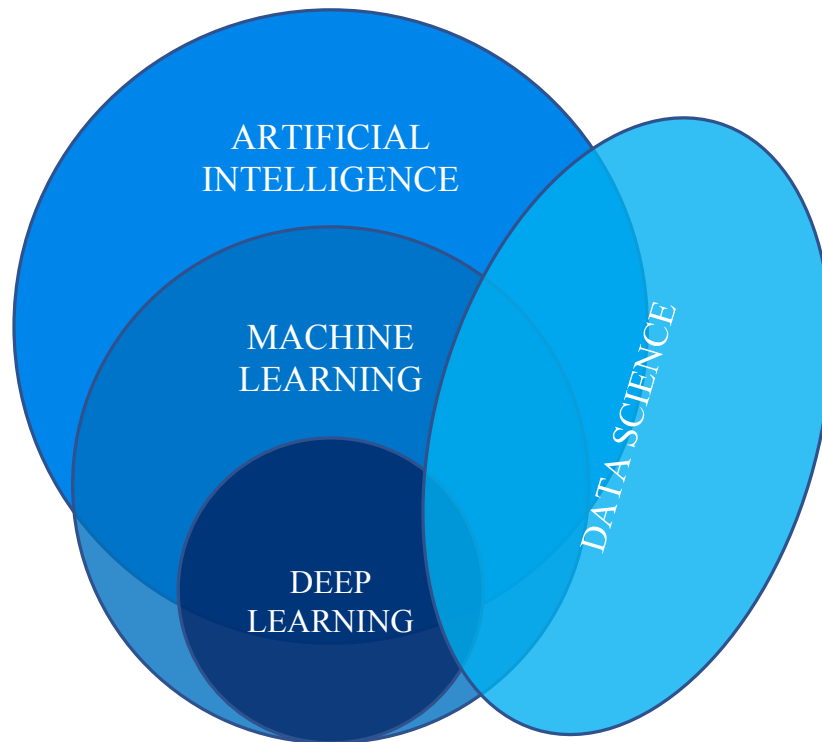
## 2.3 What is Data Science?

There are numerous terms related to data science as a field that are ill-defined, or at the very least are commonly conflated with one another. A few of these terms include *data science*, *artificial intelligence* (“AI”), *machine learning* (“ML”) and *deep learning*. Even when these are defined, there might be conflicting notions out there as these fields are constantly changing. Referring to Figure 2.1, we see overlap between these fields, but they are indeed not the same.<sup>1</sup> Broadly speaking, we will use the term AI to mean any machine that somehow mimics a portion of the human capacity of intelligence. A very important field that can accomplish artificially intelligent tasks is machine learning. Here, we focus on building the overall architecture of how we would like the system to think, and the system itself pins down the exact details. Mathematically, we construct a function with unknown parameters and additionally define a goal for that function. By feeding the system data, the system chooses for itself the optimal set of parameters given the goal we have defined. Within machine learning, we have deep learning which is just a class of functions that currently outperform many other learning tasks and are the state of the art for many applications. These functions happen to be neural nets with many layers, which are referred to as *deep neural nets* or *DNN’s* (see [8] for a physics motivated discussion on why these have been so successful).

We can also describe how data science interacts with other academic disciplines. In

---

<sup>1</sup>A cursory search online will likely show a different picture altogether, adding to the overall confusion.



**Figure 2.1:** In this diagram: deep learning is seen as a subset of machine learning, machine learning as intersecting AI, and data science as intersecting all the above. Relative sizes in this diagram have little meaning. If you want to pull in money, it's best to describe what you do as *AI*, since the term carries buzz.

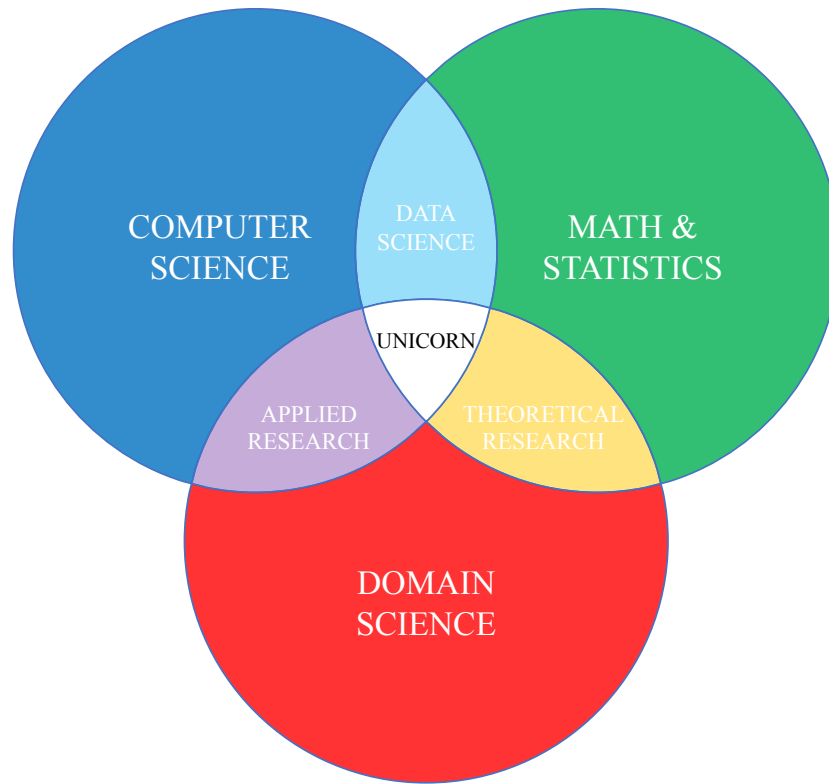
Figure 2.2 we see rough categorizations of how mathematics and statistics, computer science, and domain sciences interact with each other. Here, domain science refers to the natural sciences, such as physics, chemistry, biology, etc. Taking physics as the example of the domain science, this figure roughly describes that: the marriage of physics and math research leans more towards the theoretical realm of study, and that the combination of physics and computer science leans more toward applied studies. Researchers in all three fields can colloquially be referred to as “unicorns,” since they seem to be exceedingly rare. However, as data science broadens its realm and impact, we will likely see more unicorns around.

A few examples of “unicorn work” follow. In the domain of theoretical physics, researchers applied a convolutional neural network (“CNN”) to predict Calabi-Yau volumes from

topological quantities, thereby bypassing the standard minimization procedure that was used previously [9]. This showed the researchers that a functional relationship exists between two quantities that was not previously known, i.e., that the volume was somehow encoded in the topological information. In a different theoretical physics project, researchers constructed machine learning methods which produced conjectures in the string theory landscape that may have not been known otherwise [10]. In the field of biological medical sciences, researchers showed that retinal photographs can be used to predict various attributes of a patient, including their age, gender, potential cardiovascular risks, past cardiac events, blood pressure, and smoking status [11]. Prior to this study, it was unknown that there was a link between many of these attributes and retinas. The tool that made this link was a deep neural network.

In these examples, we see that data science can discover relationships that were previously unknown and had not even been hypothesized to exist in the first place. In mathematical terms, data science can be used to find structure we think may exist, but it can also find structure elsewhere in a place in which we didn't know to look. Take a treasure map as an analogy to this situation. The actual map itself represents the field of study, and treasure locations represent interesting results in the field. Through years of study and exploration, a physicist, for example, may be able to determine some good areas to look. Data science in this analogy would be a wondrous tool that would let the physicist know whether they're right about the locations they have predicted, and could itself also predict other spots they should look.

It is my hope that more researchers include modern data science methods into their workflows as great leaps and bounds can be made in both the applied and theoretical sciences.



**Figure 2.2:** A “not-so-quite” accurate depiction of how computer science, math & statistics, and domain sciences interact. Despite this, the takeaway is that the intersection of all three fields results in a field which is itself unique and very useful, but rare.

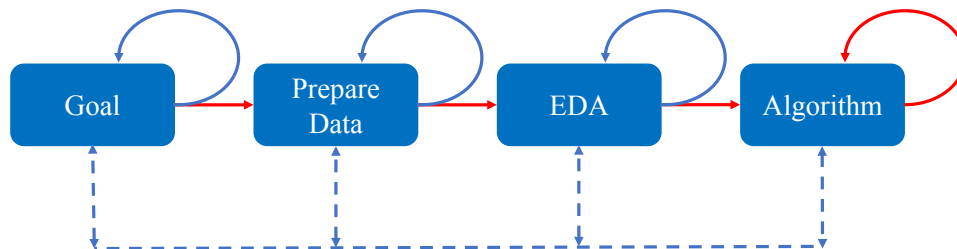
### 2.3.1 Typical Workflow

A typical workflow for a data science project is depicted in Figure 2.3 and each of these steps is described below:

1. **Goal:** The first step in this flow is to define a goal or hypothesis. In the wedding example (Section 2.2), the goal would be to define an optimal seating arrangement. Inherent in this is to additionally define a metric on how to measure how close we are to the goal. A few examples that define this metric for different projects may be: reconstruction error, classification error, precision, and recall.



2. **Prepare Data:** The next step is to decide what data we need in order to achieve the goal, while having an algorithm workflow in mind in order to construct a holistic project, and to prepare it for the data analytics downstream.
3. **Exploratory Data Analysis (“EDA”):** This critical, but sometimes overlooked step, can provide us with a wealth of information pertaining to the data. Playing around with the data in almost any way can be considered EDA, though a very powerful and common tool is data visualization. If the data is high dimensional, dimensionality reduction (Section 2.4.2) must be performed in order to visualize the data.
4. **Algorithm:** Often the last step in the workflow is to feed the data through the algorithm we have chosen to accomplish the goal. Depending on how it performs, we may have to go back to any of the other steps to refine the workflow in order to increase performance. See Section 2.4 for an overview of specific algorithms.



**Figure 2.3:** A simplified workflow of a data science project. The path in red is a “typical” workflow, but indeed any path in general can be taken. Note the cyclic processes which will likely occur as a result of gathering more information which prompts a modification at any step in the workflow. The red loop at the end refers to modifying an algorithm’s hyperparameters.

A typical flow shown in the figure is not a hard and fast rule on how to approach data science, and indeed can be modified in any way - for instance, steps might be left out altogether or the arrangement of the tasks can be changed (as seen by alternative paths in the flow). In fact, *Goal* which is placed first in the flow, can actually come last as *EDA* may inform us on what questions to be asking in the first place.

There are numerous standardized data sets, which fortunately can reduce this workflow by removing the *Prepare Data* step. Two common examples are the MNIST and CIFAR-10 datasets:

- MNIST [12]: “Modified National Institute of Standards and Technology” data consists of handwritten images of digits. These images are grayscale and discretized into  $28 \times 28$  grids. There are  $60k$  training and  $10k$  testing images.
- CIFAR-10 [13]: “Canadian Institute For Advanced Research” data consists of pictures of common objects. These images are in color and discretized into  $32 \times 32$  grids. There are  $50k$  training and  $10k$  testing images. For this dataset, there are only 10 classes of objects (see Section 2.1 for terminology).

There are numerous other standardized datasets, each of which can be used for a different specialized task. Maintaining such databases are important for the field of data science to progress as a whole so that results can be benchmarked with one another.

### 2.3.2 Bias-Variance Tradeoff

Two related sources of errors in any predictive algorithm are *bias* and *variance*. Bias refers to errors coming from the model<sup>2</sup> being too simple, hence it will miss relevant mathematical structure or links in the data. Variance refers to errors coming from the model being too complex, and hence it will construct mathematical structure or links in the data that should not be there; these links can vary greatly if the training set is changed. In other words, models that exhibit high bias are *underfit* and those that exhibit high variance are *overfit*<sup>3</sup>. Note that when we say the model is too complex, for instance, we really mean that the model is too complex with respect to the data at hand - both model and data determine whether the problem is underfit or overfit.

The “bias-variance dilemma” is the problem of minimizing both variance and bias, given there is a tradeoff between the two. Figure 2.4 captures this tradeoff and dilemma as a plot of

---

<sup>2</sup>*model* and *algorithm* will be used interchangeably

<sup>3</sup>these terms will also be used interchangeably

training and testing errors with respect to complexity. In Chapter 5, we will see a topologically based novel approach to resolve this dilemma for image classification; in Section 2.4.5 we will see how topology can resolve this dilemma more generally.

The two regimes for complexity are overfit and underfit, each of which are characterized by both model and data considerations:

### I. Overfit regime

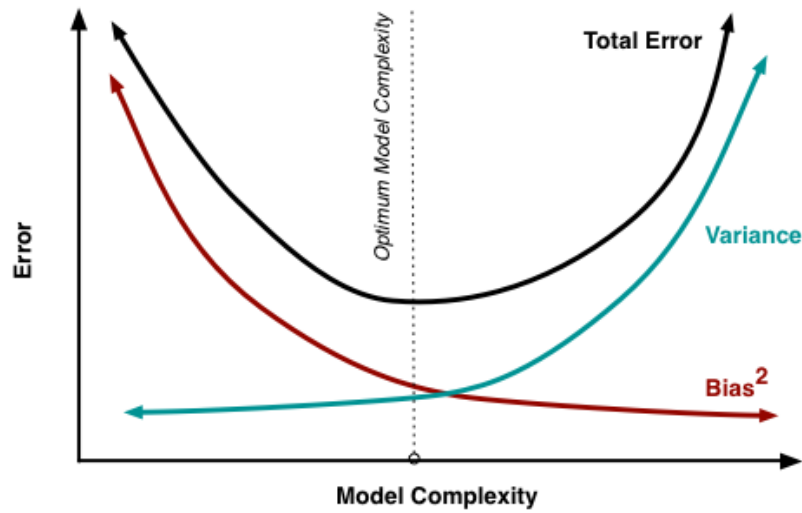
- If our model is too powerful (i.e., too many parameters)
- If we have too many features in the data (i.e., our model can distinguish every specific instance)
- Not enough data relative to model (i.e., we'll be able to fit the data perfectly well)

### II. Underfit regime

- If our model is not powerful enough
- If we don't have enough features in the data

Modern methods typically have increasingly high model complexity, especially in deep learning, so they are pushed towards the overfit regime. For instance, researchers showed that deep neural networks are capable of fitting to noisy data perfectly well (i.e., with 100% training accuracy) [15]. The theoretical reason that allows for this to happen comes from the universal approximation theorem [16, 17, 18] which states that deep neural networks can approximate any continuous function arbitrarily well.

There are various methods to resolve this bias-variance dilemma in general. Two data-oriented approaches are *bootstrapping* [19] and the *random subspace method* (also referred to as *feature bagging*) [20]. In the former case, we randomly sample which instances from the data to include in training, and in the latter case we randomly sample which features from the data



**Figure 2.4:** The bias-variance tradeoff depicted by errors as a function of model complexity. The red and blue curves represent errors due to variance and bias, respectively. The total error represents the expected generalization error, i.e., the error we would expect to get on a test data set. If the algorithm resides on the left side of the optimal complexity, boosting can help push the model to the right; if the algorithm resides on the right side, bagging can help push it left. Image from [14].

to include while training (in either case, the random sampling is done with replacement). Each of these approaches can be used in conjunction with an ensemble learning technique, which is a process by which individual algorithms are combined into one final algorithm. Bootstrapping keeps our final model from learning the individual training set too well as each individual algorithm was trained on a random subset of the data, and feature bagging reduces the correlations between the individual algorithms. These correlations can emphasize mathematical structure which is not descriptive of the data - in other words, these correlations can lead to overfitting.

Two model-oriented approaches to resolving this dilemma, which are both ensemble learning techniques, are *bagging* (the combination of “bootstrapping” and “aggregating”) [21] and *boosting* [22]. In both approaches, we construct many individual models and combine them in some way to produce one final model that has reduced variance and reduced bias. In bagging, we take individual models that would ordinarily overfit the data by themselves. Variance is

reduced by training each of the individual models on the bootstrapped data then averaging out the prediction made by each of these models. The variance can be further reduced by feature bagging. In boosting, we construct the final model from individual models that would ordinarily be underfit. Boosting is an iterative procedure whereby an additional weighted model is constructed in order to fix failures of the previous model (more on this in Appendix A.4.1).

Combinations of bagging, boosting, and feature bagging procedures commonly appear in state-of-the-art methods. In Chapter 5, we will see a similar approach used in topological data analysis in order to produce a final model with reduced variance and bias.

### **2.3.3 Ethics**

To effectively integrate data science and machine learning into the world, it is important to discuss how our algorithms currently interact with humans and how we would like this interaction to develop in the future. Both policy and ethics considerations should be included in this discussion. However, this can be difficult as data scientists are typically more interested in the algorithms themselves and policy makers typically don't understand the basics of data science. There are examples of work being done in this area, for instance at the Stanford Data Science Initiative [23] and Coursera [24], but these kinds of discussions can get pushed to the background quickly. Some important topics here pertain to: data privacy issues, biased algorithms (for instance, racially or gender biased), the intent of the algorithms, and the scope of the algorithms in terms of what they can and cannot accomplish.

## **2.4 Algorithm Classes and Techniques**

In the remaining few sections of this chapter, we will briefly discuss the main mathematical concepts used in this dissertation, which include: optimization, dimensionality reduction, clustering, classification, and topological data analysis. In-depth treatments of these topics can be

found, to an extent, in later chapters and in listed references.

## 2.4.1 Optimization

Optimization generally refers to the process of minimizing a cost function (also referred to as a “loss” function), or equivalently maximizing one, by way of searching the parameter space. There is an additional set of parameters, called hyperparameters, that involve user input and can also be considered as part of the overall optimization process. In Figure 2.3, part of the red loop at the end includes modifying these hyperparameters in order to improve the quality of the algorithm. Both the general and explicit forms of the cost function can change depending on the algorithm architecture and, more specifically, the task we’re considering. Due to this, its mathematical properties can change, which most importantly include its convexity and convergence properties. General optimization tasks can be  $\mathcal{NP}$ -hard, so it is no wonder these techniques themselves are heavily studied [25]. In Appendix A, we introduce a novel approach of selecting optimal optimizers for a class of constrained optimization tasks called mixed integer programs.

For differentiable loss functions, we can use gradient descent based techniques [26] to converge to local minima. In the case where these losses are *convex*, the local minima are guaranteed to be the global minima; in the case where they are not convex, there are numerous methods that help the optimization procedure converge to the global minimum rather than one of the local minima [27, 28]. In these gradient descent methods, we update parameters of the cost function by computing its derivatives, i.e.:

$$\theta_{new} = \theta_{old} - \alpha \nabla f(\theta)|_{\theta_{old}} \quad (2.1)$$

Here,  $f : \Theta \rightarrow \mathbb{R}$  refers to the total loss function which takes parameters  $\theta \in \Theta$  as input and computes a number.

An entirely different set of minimizing techniques is presented in Appendix A. These

techniques differ in that they handle parameters that can take on both integer and continuous values, and that have high dimensional bounds imposed on them. Due to their complexity, we often times don't know a priori which technique might be best for solving an incoming optimization problem. In Appendix A we address this issue by constructing a learning algorithm that makes this choice.

## 2.4.2 Dimensionality Reduction

Dimensionality reduction is the task of representing high dimensional manifolds with lower dimensional ones which either approximate the original space reasonably well (intuitively, think minimizing reconstruction error) or just helps us capture some other useful information about the dataset. For instance, dimensionality reduction techniques can be used to extract the most relevant features from the data; the features that are chosen depends on both the model and the lower dimension we choose for the projection. Techniques in dimensionality reduction will be discussed more in Chapters 3, 5 and Appendix A. Mathematically, dimensionality reduction is a map:

$$\phi : X^n \rightarrow X^m, \text{ where } n \geq m \quad (2.2)$$

Dimensionality reduction is important for numerous reasons, one of which is that the high dimensionality of many datasets is often times unnecessary and in fact can lead to an overfit model (Section 2.3.2). MNIST data, which consists of images of normalized and centered handwritten digits, is originally  $(28 \text{ pixels}) \times (28 \text{ pixels}) = 784$  dimensions. Just in terms of the curse of dimensionality (Section 2.2.2), we would minimally need  $\sim 2^{784} \sim 10^{236}$  points to fill the space. This is compared to the  $\sim 10^{80}$  atoms in the universe. Of course, we expect there to be a lower dimensional structure which captures all the useful information in the handwritten digit.

Two choices inherent to the dimensionality reduction procedure are the dimension we project to, i.e., the value for  $m$ , and the method we use to project the data there. Most methods can be characterized into either linear or nonlinear projections.<sup>4</sup> Some common examples of both these methods include:

### I. Linear Projections

- Principal Component Analysis (“PCA”) which maximizes the variance of the projected data [29]. This will be discussed further in Chapters 3, 5 and Appendix A.
- Fisher’s Linear Discriminant Analysis (“LDA”) which attempts to find linear combinations of features in order to classify the data [30].

### II. Nonlinear Projections

- Kernel PCA which uses the kernel trick before performing a normal PCA [31].
- An autoencoder which is a neural network architecture typically optimized to minimize reconstruction error [32]. This will be discussed further in Chapter 5.
- Multidimensional scaling (“MDS”) which attempts to preserve pairwise distances in the lower dimensional space [33]. This will be discussed further in Appendix A.
- Isomap which attempts to preserve geodesic manifold distances between all pairs of points [34].

Choosing the model type is motivated from the specific data science question at hand; the dimension  $m$  can be selected in different ways for the different models. A common method is to vary the value of  $m$  and to determine how this changes some overall metric. In the case of PCA, we can calculate the percentage of variance described by the projections to each of the choices for  $m$ , and determine for which  $m$  this quantity flattens out. This will be discussed more in Chapters 3, 5 and Appendix A.

---

<sup>4</sup>There are additionally some approaches that utilize both linear and nonlinear techniques in order to construct a single projection map



### 2.4.3 Clustering

Clustering refers to the general process of assigning instances to groups based on some affinity measure. Here, the term “groups” is used synonymously with “clusters.” Many clustering methods fall within four categories which are each determined by what this affinity measure is:

- I. *Hierarchical Clustering* constructs a family of different clusterings based solely on the distance information between points. A single clustering from this family is determined by a thresholding parameter and the family itself is constructed by varying this parameter.
- II. *Density Based Methods* define groups based on regions of higher densities in the data.
- III. *Distribution Based Methods* use probability distributions to model the data. Each distribution defines a group, and an instance is assigned to a group based on maximal probability.
- IV. *Centroid Based Methods* attempt to find the centers of groups of data (called *centroids*). Each instance is assigned to the group that contains the nearest centroid.

The approach to *hierarchical clustering* is motivated by the idea that nearby points should be categorized in the same group, however it does not specify a priori what “nearby” actually means. Hence, the family of clusterings is parameterized by a nearness parameter. As an example, *single-linkage clustering* [35] starts with  $n$  clusters given by each of the  $n$  data points. For each of these points, we can also assign a neighborhood radius  $\epsilon$ , a nearness parameter. Two existing clusters merge if they are within a distance of  $2\epsilon$  of one another (here, the distance is taken as the minimum between all possible point pairs in each of the clusters). In this method, we see how these clusters change as a function of  $\epsilon$ . Because of this, we do not choose how many clusters to group the data into before running the algorithm - this number needs to be chosen after looking at the family. Single-linkage clustering will be used in Chapters 4 and 5 to construct persistent homology diagrams and mapper objects from TDA (see Section 2.4.5), respectively.

Since *distribution based clustering* learns probability distributions, we can use these as generative models once they have been trained. A common approach is to utilize a Gaussian mixture model (“GMM”) [36], which fits the data with multiple Gaussian distributions. In Chapter 4 we will see an application of GMMs with persistent homology (see Section 2.4.5).

A common approach within *centroid based methods* is  $k$ -means clustering [37]. In this method, we choose the number of clusters prior to running the algorithm. If we choose  $k$  clusters, the algorithm will partition the data into  $k$  distinct regions called Voronoi cells [38], each of which is described by the centroid of the data.

Regardless of the clustering method we choose, an important step in the workflow is to determine how many clusters best represent our data. In some cases, we run the algorithm first and then determine the appropriate number of clusters based on its output (i.e., for hierarchical clustering). In other cases, we insert this number prior to initiating the algorithm, and iteratively see how this choice affects the outcome. In either case, there are tools to help us ultimately determine an appropriate value.

For instance, in the case of  $k$ -means clustering, we can construct “elbow plots” [39] to guide our choice, but they can sometimes be ambiguous to interpret. In Chapter 3 and Appendix A, we will see specifically how elbow plots can be utilized to determine the true dimensionality of data when used in conjunction with dimensionality reduction techniques. As another example, the G-means algorithm [40] determines the appropriate number of clusters by seeing how many Gaussian distributions are needed in order to sufficiently cluster the data. Once this number is computed we can insert it into another algorithm, for instance it can become the  $k$  value in  $k$ -means clustering.

An interesting issue with many clustering methods in high dimensions stems from the curse of dimensionality. Under certain conditions, as the number of dimensions increase, the distinction between the nearest points and farthest points in the data gets washed away [41]. Fortunately, when we expect the data to occupy fundamentally lower dimensional manifolds,

clustering in high dimensions can still make sense.

Mapper objects from TDA, which will be used in Chapter 5, are related to existing methods in clustering, but they don't quite fit neatly into any one of the four above-mentioned clustering methods. See Section 2.4.5 for an introduction to these objects and Chapter 5 where they will be used to construct robust classifiers.

## 2.4.4 Classification

Classification is a very common task in machine learning that attempts to find a function  $f$  that maps features of an instance to its predicted label. More specifically, this function  $f$  can be thought of as a map:

$$f : X \rightarrow Y \tag{2.3}$$

Here,  $x \in X$  refers to a vector of features for a specific instance and  $y \in Y$  refers to the possible labels.

There are two approaches to constructing the map  $f$  which are referred to as “eager” and “lazy” learning. In eager learning, we attempt to model a probability distribution from the training data underlying this mapping. The learned distribution differs slightly based on whether we take a discriminative or generative approach:

$$\begin{aligned} f(x) &= \underset{y}{\operatorname{argmax}} p(y|x), \text{ discriminative} \\ f(x) &= \underset{y}{\operatorname{argmax}} p(x|y)p(y) = \underset{y}{\operatorname{argmax}} p(x, y), \text{ generative} \end{aligned} \tag{2.4}$$

Using Bayes' Theorem [42], we see the two maps  $f$  are equivalent, though the learned distributions differ. Geometrically, discriminative classification models can be thought of as drawing hypersurfaces in  $X$  in order to separate the labels in  $Y$  as best as possible. These surfaces are referred to as “decision boundaries.” Generative models carry more information, and indeed

we can infer these boundaries from them as well. When an unseen test point is run through an eager classification model, it is mapped to the label with the highest probability. This of course works well when the trained probability distribution also represents the testing probability distribution.

In lazy learning, we store the training data and don't learn a model beforehand. Instead, given the unseen test point, we classify it based on its relatedness to existing training points in  $X$ . Throughout this dissertation, we will see both approaches utilized.

Two common eager classifiers are neural networks [43] and random forests [44], and a common lazy classifier is  $k$ -nearest neighbors (" $k$ -NN") [45]. Neural networks have the ability to draw any arbitrary decision boundary, owing to the universal approximation theorem [16]; because of this they are prone to overfitting. Random forests draw linear decision boundaries and are typically less prone to overfitting. For  $k$ -NN, we assign a label to a test point by looking at the  $k$  nearest neighbor's labels in  $X$ . For  $k = 1$ , 1-NN draws the conventional Voronoi boundaries [38] between points [46].

## 2.4.5 Topological Data Analysis

In this final section of the chapter, we will briefly review TDA and its motivations from a mathematical perspective, its motivations from a human based learning perspective, and a literature review. Later on in Chapters 4 and 5, we will see specific applications of persistence and mapper from TDA.

### Mathematical Motivation

Many machine learning methods are fundamentally built upon notions from geometry: including length, area, volume, local curvature and others. These geometric concepts can be thought of as inferring the structure of the data by building up from local information. This usually means that if we perturb the data in some way or change some of the parameters in the

workflow, the outcome of the algorithm may completely change. If, on the other hand, we can infer more global information such as the structure of the data or the number of holes in the data, our algorithm may be more robust to these types of changes. This is the type of information topological data analysis attempts to capture.

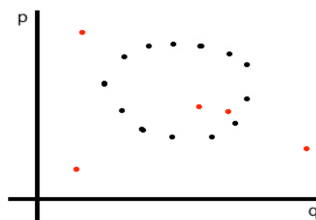
Topological data analysis (“TDA”) is a relatively new field of data science, but is motivated by much older mathematical concepts. Generally speaking, it wishes to extract topological quantities from data, which tend to focus on the broader global structure of the data rather than local information. The tools that allow us to do this come from algebraic topology [47] and, in the case of TDA, have been extended to allow for the investigation of discretized data spaces. There are numerous reasons to use TDA in a data science pipeline, many of which come from three fundamental properties inherent to the field:

1. Compressed Representation
2. Coordinate Invariance
3. Deformation Invariance

*Compressed representation* can be thought of as a shortened summary of the data that has been distilled down to its most important structure. Owing to this, TDA is often used solely as an EDA tool since we can visualize this distilled structure. While this is an important step in the data science workflow (Figure 2.3), it is by no means the only place TDA can or should be used. In this dissertation, we will see TDA being used at other steps in the workflow - sometimes it will be useful to think of TDA as being part of the data preprocessing and at other times it will be more useful to think of it being fundamentally built into the algorithm. However we think about it, placing TDA in other workflow spots can be useful especially if we would like to incorporate *coordinate invariance* or *deformation invariance*, which are particularly important in computer vision (Chapters 4 and 5). We will see in Chapters 4 and 5, that although TDA is



(a) A dataset that is topologically equivalent to a circle.



(b) A dataset that is topologically equivalent to a circle, despite the added noise represented by the red points.

**Figure 2.5:** Both of these datasets are topologically equivalent to a circle. TDA in general attempts to capture this information.

theoretically motivated by these three properties, in practice the TDA method we use may have different regimes on how strongly these properties are met.

The two tools in this dissertation that will be used from TDA are called *persistent homology*, or *persistence* for short, and *mapper* graphs, or just *mappers*. Refer to [48, 49, 50] for overviews of persistent homology and [7, 51, 52] for overviews of mapper.

## Human Based Learning Motivation

In Chapter 5, we will see mapper objects being used as part of the algorithm step in the workflow (Figure 2.3) in order to produce a robust classifier. This approach can be thought of as a means to minimize the generalization error due to variance (see Section 2.3.2), since it attempts to capture global structure of the data. In other words, we produce an underfit model in this approach due to its coarse graining, but we regain complexity by utilizing an ensemble approach in conjunction with conventional classifiers - i.e., we start out to the left in Figure 2.4 and move more right. It appears that humans also resolve the bias-variance tradeoff by starting out left then moving right [53] in a “less-is-more approach.” TDA allows for generality in another way as well: we can use TDA when we only have a notion of similarity in the data.

Neural nets and random forests, on the other hand, typically start out to the right in Figure 2.4 and move more left. Additionally, these methods typically need very strong notions of

similarity in the data. Because of these features, these algorithms can perform much worse than humans do at certain tasks. Some issues specific to neural networks include:

- The ability of convolutional neural nets to perfectly fit random noise [15], which is theoretically consistent with the universal approximation theorem [16, 17, 18]
- The existence of adversarial examples for neural nets, which can be thought of as minimally intrusive ways to trick the nets [54]
- The necessity of many data points in order to train the nets
- The computational intensity of training the nets
- The difficulty in optimizing the nets

These issues point to the fact that, although deep learning and ML methods have been widely successful, they process data differently than humans do.

## Literature Review

Data science is a rapidly changing field and TDA is no exception. Here, we will see a brief overview of past and current accomplishments in TDA. Additional citations will be given where relevant in later chapters.

### *1. TDA as an EDA tool*

TDA methods are commonly used as EDA tools since they produce low dimensional visualizations of the data. Persistence has been used to show that natural images are topologically equivalent to the Klein bottle, and that this correspondence can be used as an encoding scheme for images [55]. In [56], persistence is used to drive the hypothesis that certain activations in the visual cortex correspond to certain topologies. In [57, 58], persistence is used to study galaxy distributions, which construct the “cosmic web.” In [59], mapper objects are used to study RNA hairpins to identify dominant folding paths. In [60], mapper objects are used to:

discover significant clusters in breast cancer gene expression, classify NBA player performance, and uncover voting behavior in the United States Congress.

## *2. Well Definedness of TDA*

There are numerous rigorous mathematical studies which show that methods in TDA are well defined, and that they maintain coordinate and deformation invariance under certain assumptions. In [61], persistence is used to study how the homology of randomly generated data changes with respect to the choice in distributions used to create the data, and in [62] the same authors establish a connection between probability theory and homology. Sampling and noise were studied in [63], which provides bounds on how many samples need to be drawn from a dataset in order to accurately recover its homology, and how much noise can be added in order to recover the homology. The “nerve theorem” [64] guarantees that the simplicial complex used in persistence is homotopy equivalent to the underlying space. [65] shows barcodes are stable for perturbations in a Morse filtration for computing persistence, and [66] provides stability guarantees on mappers and shows that they are structurally a coarse graining of Reeb graphs. [67] presents robust and simplified methods (in terms of complexity) for computing persistence.

## *3. ML with TDA*

The utility in using TDA comes from the three properties listed in Section 2.4.5, namely compressed representation, coordinate invariance and deformation invariance. When we use TDA along with well established ML methods, which typically don’t have these properties, we are able to construct a more robust algorithm. In [68], mappers are used to classify error modes for CNNs when applied to MNIST data, and are consequently used to enhance classification robustness. In [69], persistence is leveraged as a regularizer in a classifier. In [70], metrics are learned in order to calculate distances between persistence diagrams; these are then used in order to classify graphs. In [71], persistence is used as a graph reconstruction tool in noisy data.

## *4. The Software of TDA*

There are several different open source implementations for computing mappers: TDAmap-



per [72] (implementation in R), Python Mapper [52] (implementation in Python), KeplerMapper [73] (implementation in Python).

There are also open source packages for computing persistence, which include: JavaPlex [74] (MATLAB), Perseus [75] (C++), Dionysus [76] (C++), DIPHA [77] (C++), GHUDI [78] (C++).

### *5. Other*

In [79], cohomology is used to construct an optimal ranking system and is applied to ranking titles hosted on Netflix. An alternate ML method that is related to mapper is spectral clustering. In [80], mapper is seen as a generalized method of spectral clustering and is used to detect communities in graphs.

# Chapter 3

## Dimensionality Reduction on Congress

A common task in data science is to find a low dimensional representation of the data in question. Doing so might help fix issues stemming from the curse of dimensionality (Section 2.2.2). It can also help us pick out which features in the data are the most important descriptors of the data or are the most useful for tasks down the line (Figure 2.3), for instance in a classification task. Additionally, dimensionality reduction can be used to visualize the data which is a very powerful tool in exploratory data analysis (Section 2.3.1).

This chapter introduces Principal Component Analysis (“PCA”), a linear dimensionality reduction technique, as an application to studying the true dimensionality of a dataset. In Chapter 5 and Appendix A we will additionally see applications of nonlinear techniques. See Section 2.4.2 for a short overview of these methods. By using PCA, we will investigate low dimensional representations of voting behavior in the United States Congress. This data is referred to as *roll call data* or simply the *roll call*. Singular Value Decomposition (“SVD”) is an equivalent dimensionality reduction technique (see Section 3.3), and we will refer to PCA and SVD interchangeably.

There are numerous methods for choosing optimal low rank approximations of data by using PCA. The optimal rank is equivalently the optimal dimension of the lower dimensional

manifold, and we refer to this optimal dimension as the “true” dimension. For instance, [81] and [82] estimate rank through Factor Analysis and Principal Component Analysis, respectively. For more recent studies, refer to [83, 84, 85, 86, 87]. In this analysis, we apply results from [88] to find the optimal dimension for a low rank approximation of the roll call data. A distinct but similar concept we investigate is the extent of the cumulative explained variance contained in a strictly one-dimensional representation of the roll call data. According to Poole and Rosenthal [89], roll call data is increasingly better explained by one-dimensional representations, especially in recent history. We take a different approach to that seen in Poole’s and Rosenthal’s analysis, and we come to a more fine-grained result. For additional background in political polarization in the United States, refer to [90, 91].

### 3.1 Data

The data we use comes from Keith Poole’s and Howard Rosenthal’s Voteview project [92], which consists of roll call data for both the House of Representatives and Senate for Congress between the 1st and 113th sessions (spanning the years from 1789 to 2015), where each session spans two years. For each session, the data is represented as an  $n \times m$  matrix where  $n$  represents the number of politicians and  $m$  represents the number of votes in the session. We point out that both  $n$  and  $m$  may change between sessions, with larger variation seen in  $m$ . The various types of votes are assigned the following integers in this data:

- 0 Not a member of the chamber when the roll call vote was taken
- 1 Yea
- 2 Paired Yea
- 3 Announced Yea

- 4 Announced Nay
- 5 Paired Nay
- 6 Nay
- 7 Present (some Congresses, also not used some Congresses)
- 8 Present (some Congresses, also not used some Congresses)
- 9 Not Voting

In our analysis we are interested in three classes of votes: votes in the affirmative, votes in the negative, and null votes. Thus, we make the following replacements in the data from Voteview:  $\{1, 2, 3\} \rightarrow 1$ ,  $\{4, 5, 6\} \rightarrow -1$ ,  $\{0, 7, 8, 9\} \rightarrow 0$ . This procedure is similar to that followed in [86].

There are numerous studies in how to best choose low rank approximations in roll call data. For instance, Poole considers how well the W-NOMINATE algorithm performs in classifying correct voting patterns as a function of rank [89]. If the classification performs sufficiently well for a certain dimension, then the data is said to have a low-dimensional approximation equal to this dimension. Our approach differs in that we compute this dimension directly, without assuming anything about how well a classification scheme should perform in order to classify the data as a certain dimension.

## 3.2 Summary of Results

The main results of this chapter are summarized here. The first result is consistent with other work [89]; the second result is new to the best of our knowledge.

1. We construct the cumulative explained variance from PCA in order to:

- See how this quantity flows over time for Congress. This trend shows us that a one-dimensional PCA model has increasingly become a better approximation of the data since 1969 in the Senate and 1979 in the House of Representatives.
2. We apply the main method from [88] to:
- See how the dimension of Congress flows over time. This relative trend shows us that the overall dimensionality of the data has been increasing since 1979 in both the Senate and House of Representatives. We interpret this trend as meaning the number of political factions is increasing. This result may appear to be in direct conflict with the previous result, and we resolve this paradox later.

### **3.3 Principal Component Analysis and Singular Value Decomposition**

Principal Component Analysis (“PCA”) is a method that constructs a new coordinate system to represent data. This new coordinate system is constructed to maximize the variance in data amongst the axes, under the constraint that each axis is orthogonal. The data represented in the first principal component, or first new axis in the coordinate system, has the largest variance. Each additional axis describes less variance of the data. PCA can be used as a dimensional reduction tool owing to this decrease in descriptive power (i.e., after a certain point, the series can be truncated when it reasonably approximates the data). Singular Value Decomposition (“SVD”) is a similar method, and we point out that our results remain unchanged if we were to alternatively use this approach. In PCA, one diagonalizes the covariance matrix, and in SVD the data matrix is factored directly. Below, we use  $X$  to represent an  $n \times m$  data matrix.

## PCA

$$C = X^T X / (n - 1) = V \Sigma V^T$$

## SVD

$$X = U S V^T$$

The diagonal entries of  $\Sigma$  are the variances of the data in the new PCA space, and the diagonal entries of  $S$  are called the singular values. These constants represent the same information as they are related via:  $\sigma_i^2 = \frac{s_i^2}{n-1}$ . In particular, this means that using results from SVD are equivalent so long as the appropriate substitution is made for the variances in terms of the singular values. We point out this relation between PCA and SVD for two reasons: (1) we have seen researchers in this field using PCA or SVD; (2) it may help to think about variances or singular values depending on the context. Using either of these methods, we are interested in a few questions about the data. We address each of the questions below in the remaining subsections of this section.

- How do the singular values change with respect to time, and does this say anything important about the data series?
- How does the optimal choice in rank change with respect to time, and what does this trend say about the data series?

### 3.3.1 Information Flow over Time

In this section, we answer the first question: “How do the singular values change with respect to time, and what does this say about the data series?” We use PCA to map from the

original feature space to the new PCA space:  $\mathcal{F} \rightarrow \hat{\mathcal{F}}$ . We vary the dimension of  $\hat{\mathcal{F}}$  and calculate the percentage of variance described by this space to study the information dynamic of projecting to lower dimensional spaces. We compute the “information” retained by  $\hat{\mathcal{F}}$  as:

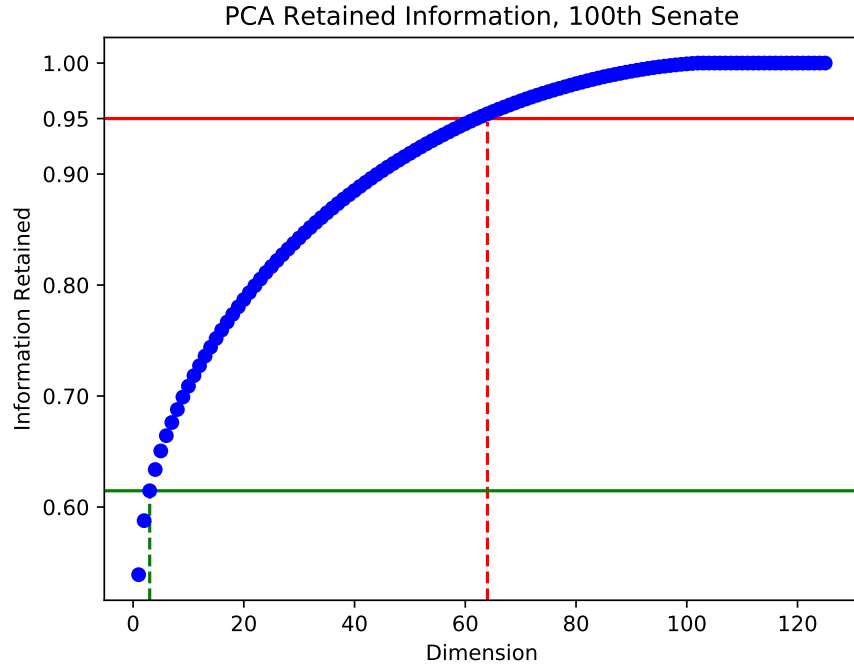
$$I(d) = \frac{\sum_{i=1}^d \hat{\sigma}_i^2}{\sum_{i=1}^m \hat{\sigma}_i^2} \quad (3.1)$$

Where  $m$  is the dimension of  $\mathcal{F}$ ,  $d$  is the dimension of  $\hat{\mathcal{F}}$ , and  $\hat{\sigma}_i^2$  is the variance of the data represented by the  $i$ -th axis in  $\hat{\mathcal{F}}$ . This ratio is typically referred to as “PCA explained variance” or “cumulative explained variance,” and we take a non-traditional approach in calling this ratio the “information” as shorthand.<sup>1</sup> Since this equation represents a percentage value, it is in the range of  $[0, 1]$  and monotonically increases with  $d$ . Additionally,  $I(0) = 0$  and  $I(m) = 1$ . See Figures 3.1, 3.2 for plots of the information as a function of  $d$  for the 110th Senate and House (years 2007-2009), respectively. It is tempting to use the information content of the data to determine the “true” dimensionality of the data by setting an information threshold then solving for the dimension that crosses this value. For instance,  $d_{\text{true}} \equiv \arg(I(d) = 0.95)$ . Although keeping 95% of the information seems an appropriate threshold to make, it is somewhat arbitrary. We address this problem later by using an alternative method which discovers values we refer to as “GD-thresholds,” coming from the authors Matan Gavish and David Donoho [88].

As has been done in [89], we aim to quantify how good of an approximation a one-dimensional model is with respect to time. To do this, we plot the trend in  $I(d = 1)(t)$ , where  $t$  represents the specific congress session start year (see Figures 3.3 through 3.6). For instance  $I(d = 1)(t = 2007)$  represents the one-dimensional information content from the 2007 to 2009 Congress. We can similarly ask how well  $d = 2$  or  $d = 3$ , and so forth, approximations are, and the reason we choose to analyze the  $d = 1$  case is due to the existing claim that a one-dimensional model can be used to accurately describe voting patterns in roll call data, especially in recent history [89]. To this end, we answer how well a  $d = 1$  approximation is with respect to time.

---

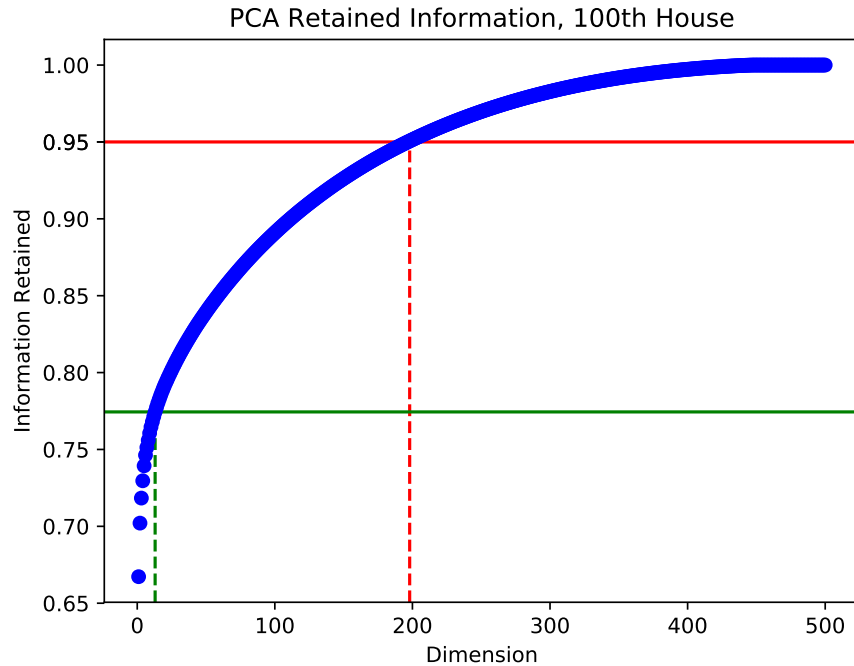
<sup>1</sup>Not to be confused with rigorous concepts of information as they apply to PCA.



**Figure 3.1:** The PCA retained information as a function of dimension given by Equation (3.1). The input data is the roll call vote for the 110th Senate (years 2007-2009). The red horizontal line defines a 95% threshold and the green horizontal line defines the GD-threshold. The vertical lines show the corresponding dimension, which in either case are much lower than the total number of votes which was 657.

To put the data from year to year on equal footing, and to produce a measure of error for the quantities we calculate, we randomly select rows and columns from the data matrix which results in a square matrix. Sampling the rows is referred to as *bootstrapping* and the columns as *feature bagging* - see Section 2.3.2 for an overview of these methods. From this matrix, we are able to compute  $I(d = 1)(t)$  and we repeat this procedure at each time step by selecting a different subset of rows and columns on each iteration, ultimately to produce an average and variance. Specifically, we produce  $50 \times 50$  roll call data for both the Senate and House, and we iterate this procedure 500 times at each time step. In general we can iterate this procedure more, for instance 1000 times, but at this point the error  $\sigma_e$  becomes arbitrarily small. By sampling the data in this manner, which puts all the data on equal footing, we are able to compare between the Senate and House and also between sessions for each branch. We use a similar sampling procedure in later





**Figure 3.2:** The PCA retained information as a function of dimension given by Equation (3.1). The input data is the roll call vote for the 110th House (years 2007-2009). The red horizontal line defines a 95% threshold and the green horizontal line defines the GD-threshold. The vertical lines show the corresponding dimension, which in either case are much lower than the total number of votes which was 1,865.

sections. Figures 3.3 through 3.6 plot this average along with the standard error (we use  $\pm 2\sigma_e$  on plots for the error). However, the errors are so small on these specific figures that they do not appear, whereas in some later figures they do appear.

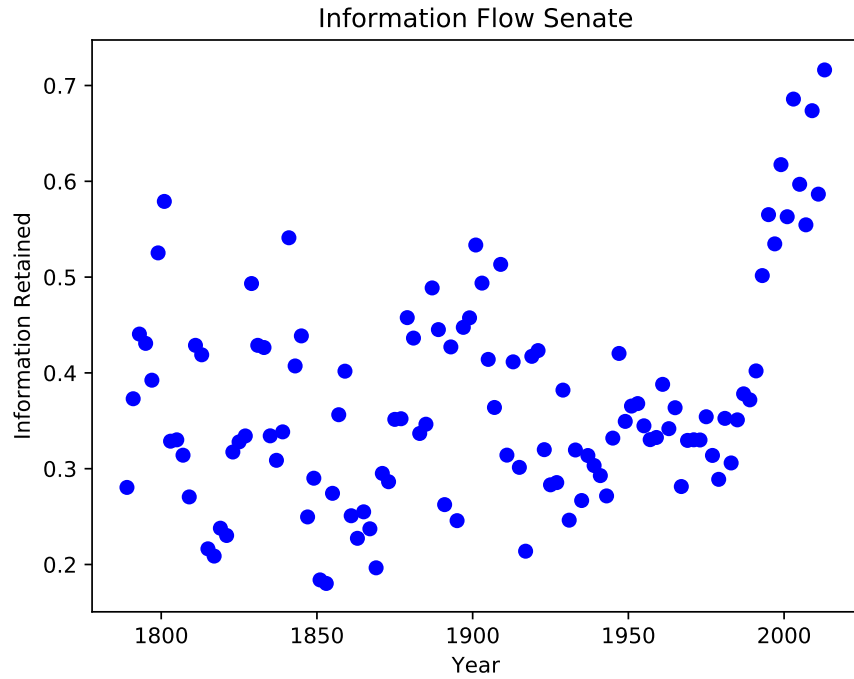
Figures 3.3 and 3.4 show the information flow for all Congress sessions from 1789 to 2015, while Figures 3.5 and 3.6 are restricted to more recent years. The clearest trends occur directly after 1979 for the Senate and 1969 for the House, in which on average, the flow is upward. This means that the first principal component is increasingly describing relatively more variance in the data. In other words, if a one-dimensional PCA model is to be used, this approximation is becoming better as this flow is upward in recent history. According to [93], parties produce low dimensional/partisan ideologies in order to win office. It is this pressure to win office that is driving this flow upward. There are three additional questions here, and we only posit their

answers. The interested reader may wish to investigate these points more.

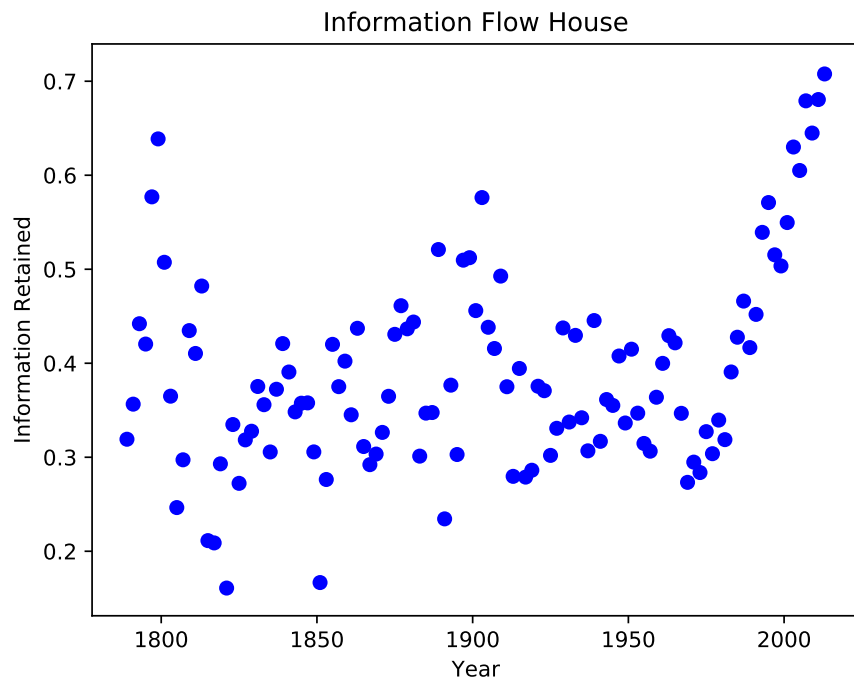
The first question is “why do these trends start to flow upwards only in recent history?”, the second is “why is there a discrepancy between when the trend becomes upward for the House and Senate?”, and finally “why is the trend for the Senate more dispersed than the trend for the House?” We posit that the increasing use of media, especially televisions, connected politicians to their constituents in such a way as had never been possible before. Owing to this increased connection, politicians were pressured to produce and present their low dimensional ideologies as in [93]. Second, the discrepancy between when the trend becomes upward for the House and Senate may be a direct consequence of the length of terms served by the politicians. Since senators’ elected terms are longer than that of house officials, they have to win office at a less rapid rate, which is the driving force ultimately producing these low dimensional ideologies according to [93]. We also attribute this constant driving force as the reason for the stability, or less dispersed results, we see in the information trend for the House when comparing to the Senate trend.

### 3.3.2 Optimal Rank Flow over Time

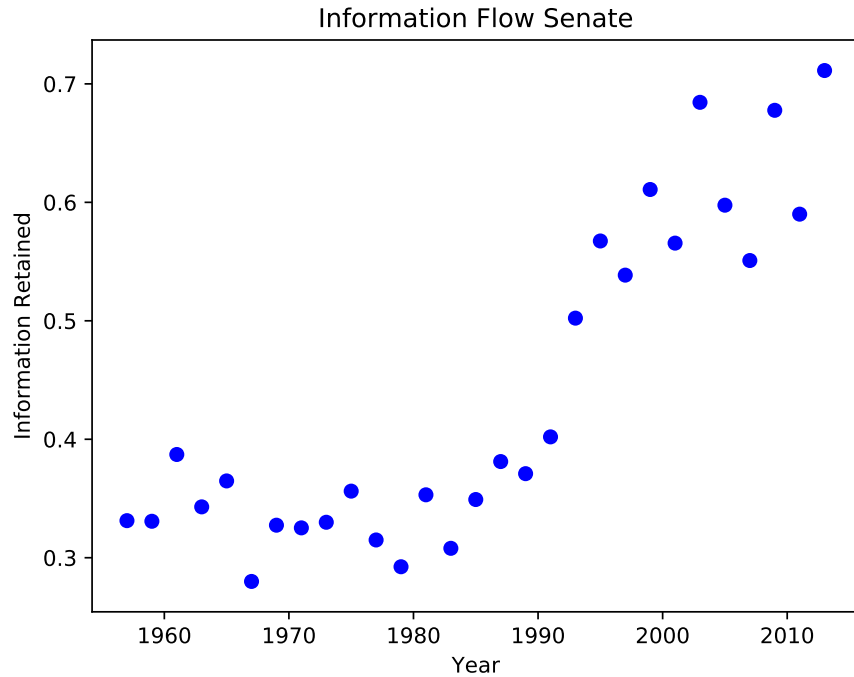
As mentioned in the previous section, the true dimensionality (i.e., optimal rank) of the data can be determined by setting a threshold in the information function (3.1). However, a more useful threshold can be constructed using results from [88]. The authors construct hard thresholds in singular values from noisy data and we refer to these as “GD-thresholds.” These thresholds are optimal in the sense that they produce an optimally low value for the asymptotic mean squared error between data (without noise) and a low-rank approximation to the data (including noise). An optimal value on this error means that the low-rank approximation to the noisy data is optimally close to the fundamental structure of the data. One interesting result we will point out is that the information approach of setting  $I(d) = 0.95$  and the GD-threshold approach are negatively correlated to one another. However, the results from the former approach appear to be



**Figure 3.3:** The flow in  $I(d = 1)(t)$  for the Senate for years 1789 to 2015 (1st to 113th Congress).



**Figure 3.4:** The flow in  $I(d = 1)(t)$  for the House for years 1789 to 2015 (1st to 113th Congress).



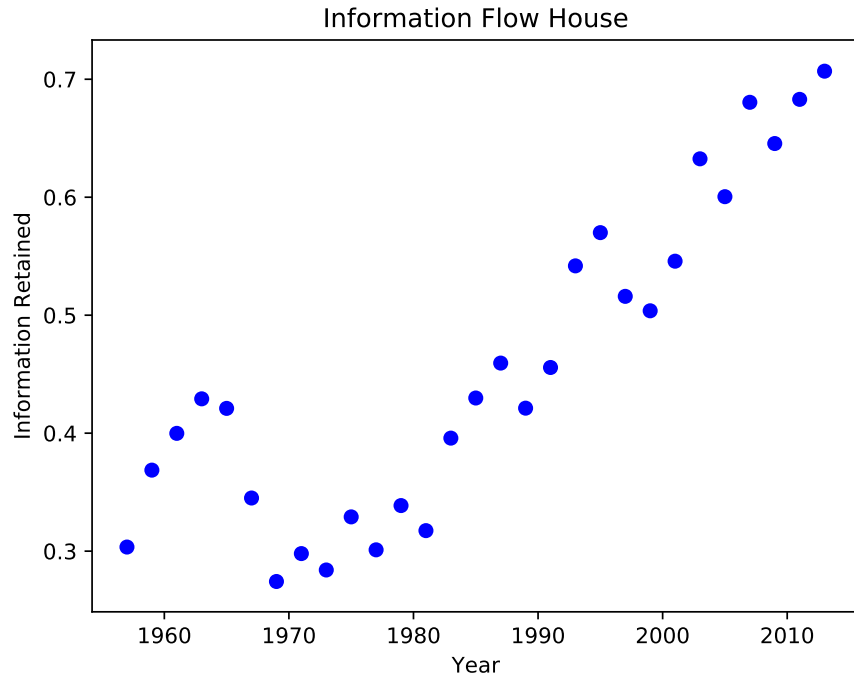
**Figure 3.5:** The flow in  $I(d = 1)(t)$  of the Senate roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1979 Congress. This figure is Figure 3.3 restricted to more recent years.

more “fuzzy,” likely due to the fact that it does nothing to denoise the data so some of the true dimensionality may be plagued by noise even if the information threshold is set somewhat low.

The result we use from [88] is that the optimal choice of hard threshold, for an  $n \times n$  matrix with unknown levels of white noise, is:

$$2.858y_{\text{med}}$$

where  $y_{\text{med}}$  is the median singular value. We define this quantity to be the “GD-threshold.” The general results in [88] apply to data with  $n \leq m$  for an  $n \times m$  matrix. Before bootstrapping and feature bagging our data, this requirement is not always satisfied. To calculate the true dimensionality of the data, we again produce  $50 \times 50$  roll call data for both the Senate and House, and we iterate this procedure 500 times for each of the time steps. While sampling the data in



**Figure 3.6:** The flow in  $I(d = 1)(t)$  of the House roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1969 Congress. This figure is Figure 3.4 restricted to more recent years.

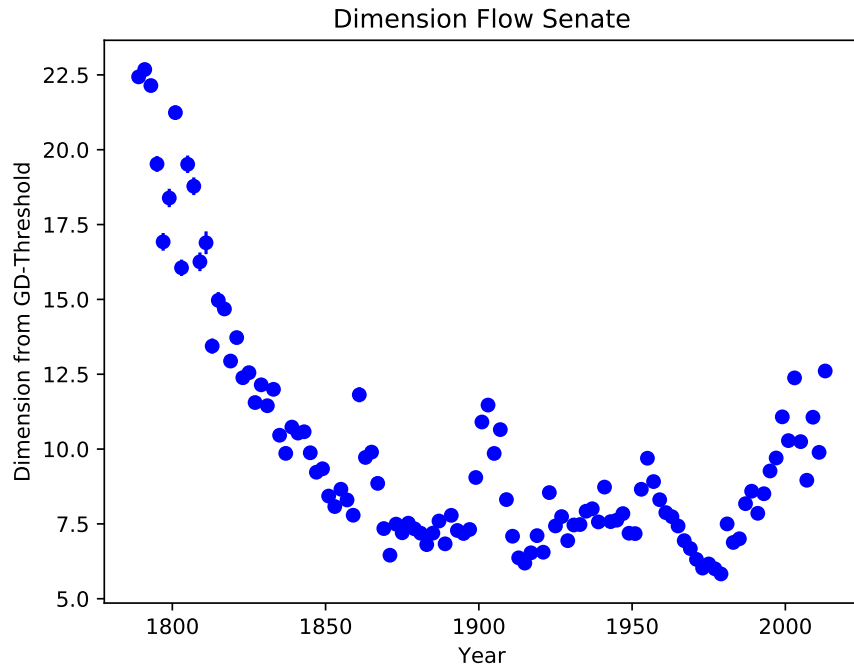
this way allows us to compare between the Senate and House, and from year to year, the actual value of  $d_{\text{true}}$  may not be significant or provide much insight since we are sampling a subset of the column space. Rather, the trend in  $d_{\text{true}}$  is what we are interested in in this section, and sampling the data in this way allows us to determine the relative changes in  $d_{\text{true}}$ .

Figures 3.7 and 3.8 show this flow for all Congress sessions from 1789 to 2015, Figures 3.9 and 3.10 show these same plots restricted to more recent years. Again, the clearest trends in these plots occur directly after the 1979 Congress in both the Senate and House, and the trend again is upward. Referring back to the information flow, namely that a one-dimensional model has been increasingly a better descriptor of the data in recent history, we are faced with the following question of how this can be consistent with an increase in the dimensionality of the data.

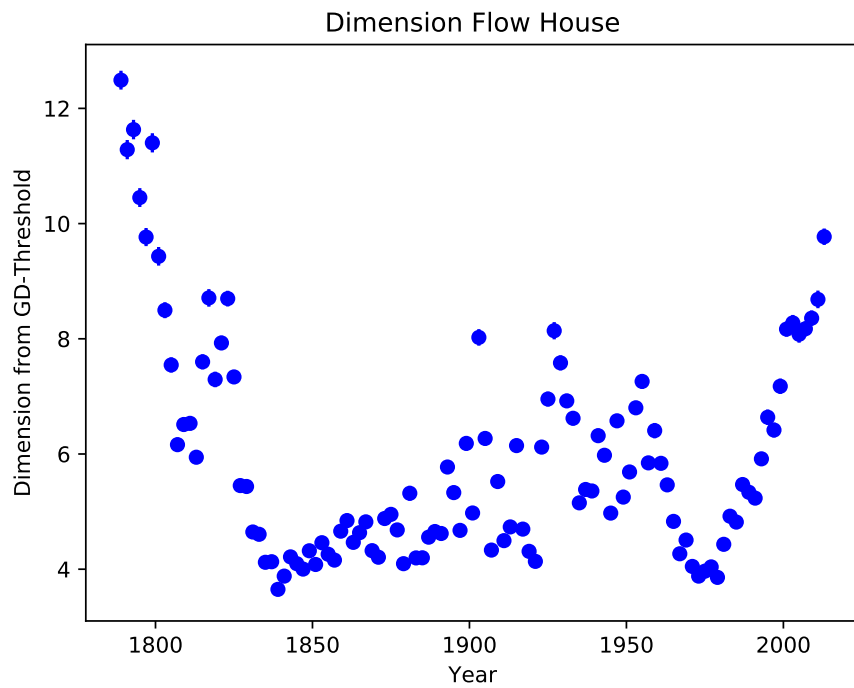
The trends from the information flow tell us that if a one-dimensional PCA model is used, this model has been an increasingly better approximation to the data in recent history. Similarly,

the increase in the dimensionality of the data tells us that more total dimensions are necessary to represent the data. Although these trends seem to be contradictory they are mathematically compatible. As a toy model, consider an ellipse in two dimensions as a descriptor of some dataset. Now, stretch the data along the first principal component to increase its variance, and also give the data a third dimension. This model describes an example where a one-dimensional PCA model becomes an increasingly better approximation of the data and with more dimensions becoming important to describe the data.

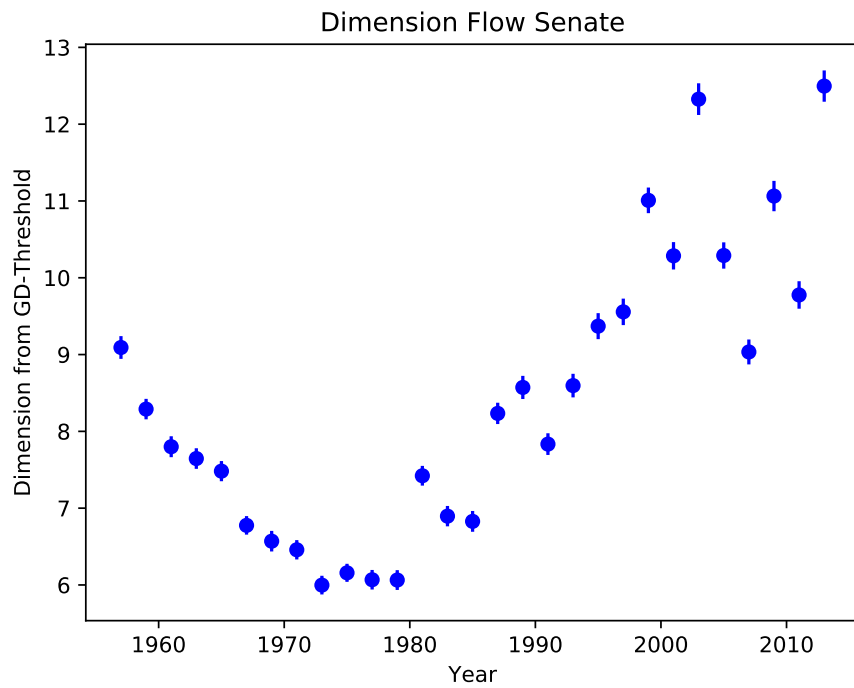
As discussed, we attribute the driving force that produces increasingly better one-dimensional PCA approximations to political parties which produce low dimensional ideologies in order to win office [93]. We posit that the driving force that has been increasing the dimensionality in recent years is linked to the number of voting blocs or to new complex voting strategies within the parties. It is reasonable to imagine that an increase in the voting blocs or new strategies would cause a “leaking” into other dimensions in the data as is the case in the toy model. Depending on how the data moves into these new dimensions, we may be able to determine the precise cause of the increasing dimensionality. New clusters appearing would provide evidence that the increasing dimensionality is due to voting blocs, and if they do not appear, the higher dimensionality may be attributed to the new voting strategies.



**Figure 3.7:** Flow in the dimension of the Senate roll call data for years 1789 to 2015 (1st to 113th Congress).

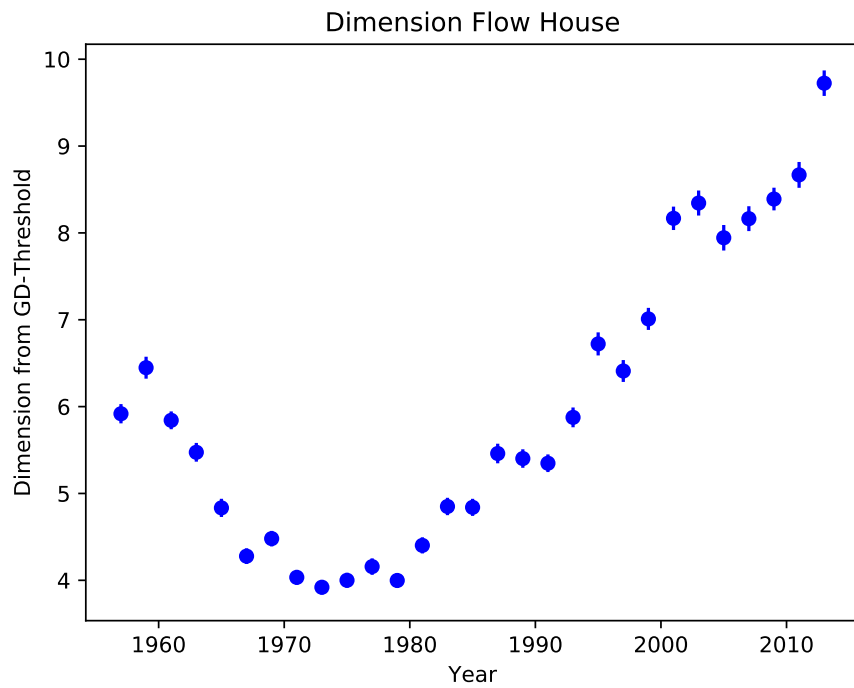


**Figure 3.8:** Flow in the dimension of the House roll call data for years 1789 to 2015 (1st to 113th Congress).



**Figure 3.9:** Flow in the dimension of the Senate roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1979 Congress. This figure is Figure 3.7 restricted to more recent years.





**Figure 3.10:** Flow in the dimension of the House roll call data for years 1957 to 2015 (85th to 113th Congress). Notice the clear upward flow after the 1979 Congress. This figure is Figure 3.8 restricted to more recent years.

## 3.4 Summary

In this analysis, we mainly focused on the upward trends seen in recent history in the information and dimension flows. Similar analyses can be applied to any time interval by looking at the combination of these flows. Additionally, we focused more on the mathematical explanations to these trends in this analysis. We leave it to the reader to discover the historical, political, or broader significance in the trends we show. To reiterate the questions we do not have answers for:

- Why do the information & true dimension trends start to flow upwards only in recent history?
- Why is there a discrepancy between when the information trend becomes upward for the House and Senate?
- What is the significance of the dispersion in these trends?

We have shown two new results in this analysis. Due to these results, we argue that Congress is not low dimensional as is often the claim, for instance in [89, 93], but rather has a richer structure than this. We summarize the trends seen thus far in both the information and dimension flows:

- House: upward trend in information starting 1969 (91st Congress), upward trend in dimension starting 1979 (96th Congress); both trends are less dispersed than the Senate trends
- Senate: upward trend in information starting 1979, upward trend in dimension starting 1979; both trends are more dispersed than the House trends

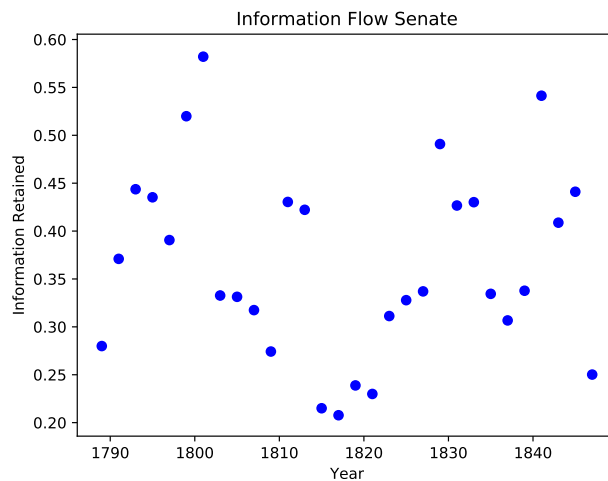
These trends show us that one-dimensional PCA models have been increasingly better descriptors of the data, and at the same time, more total dimensions are necessary to describe

the data. We attribute these mathematical trends to the increasing polarization and an increasing number of voting factions or new voting strategies in Congress. An appropriate next step for this analysis is to determine whether new clusters emerge between years, which will provide evidence of new voting blocs.

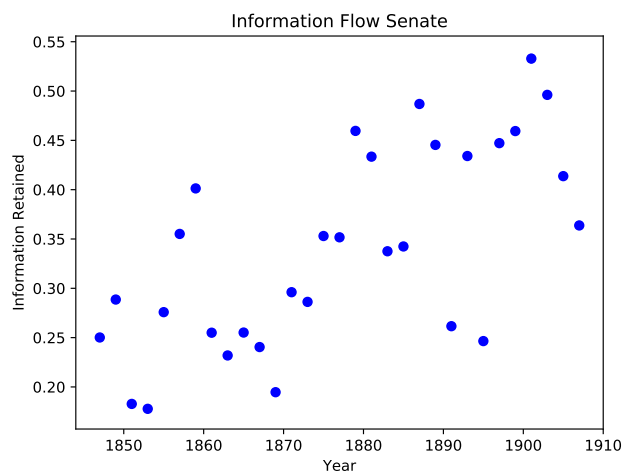
### 3.5 Acknowledgements

We would like to thank Thad Kousser at the University of California San Diego and Arthur Stein at the University of California Los Angeles for all the insights they gave into the world of political science and for the many discussions they humored my politically naive mind with.

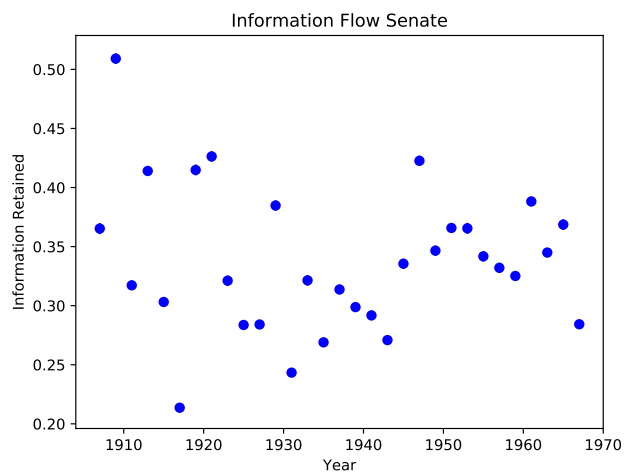
### 3.6 Additional Figures for Information and Dimension Flows



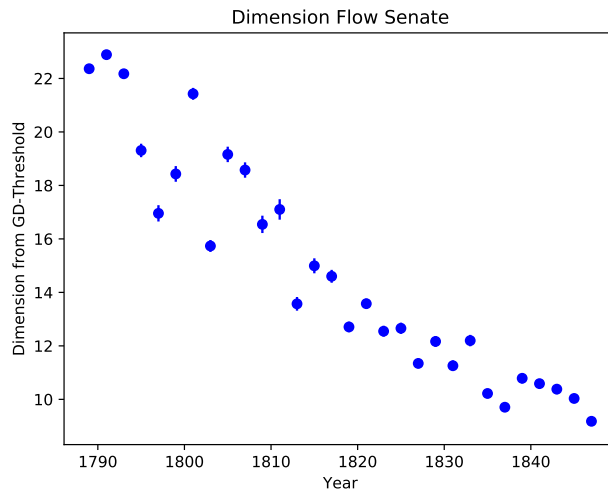
**Figure 3.11:** The flow in  $I(d = 1)(t)$  of the Senate roll call data for years 1789 to 1849 (1st to 30th Congress).



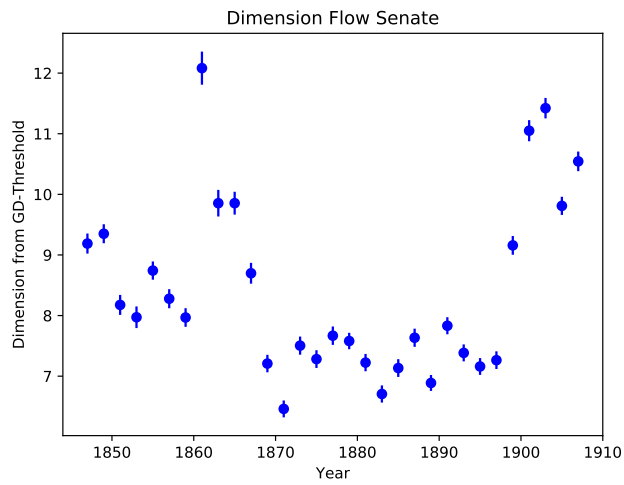
**Figure 3.12:** The flow in  $I(d = 1)(t)$  of the Senate roll call data for years 1847 to 1909 (30th to 60th Congress).



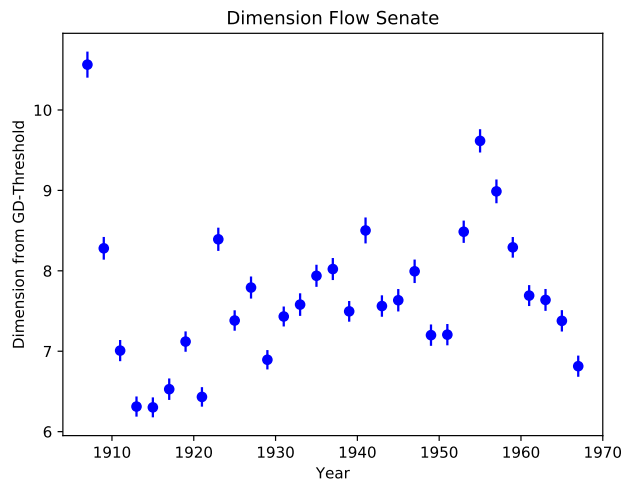
**Figure 3.13:** The flow in  $I(d = 1)(t)$  of the Senate roll call data for years 1907 to 1969 (60th to 90th Congress).



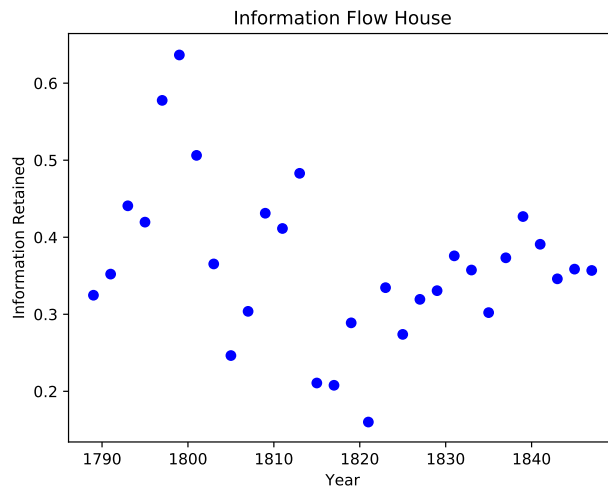
**Figure 3.14:** Flow in the dimension of the Senate roll call data for years 1789 to 1849 (1st to 30th Congress).



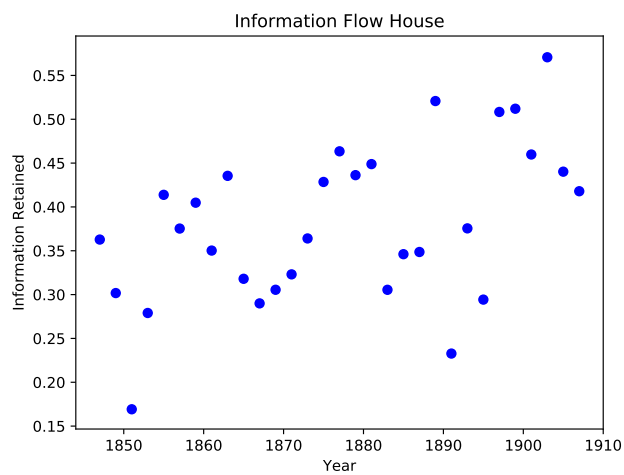
**Figure 3.15:** Flow in the dimension of the Senate roll call data for years 1847 to 1909 (30th to 60th Congress).



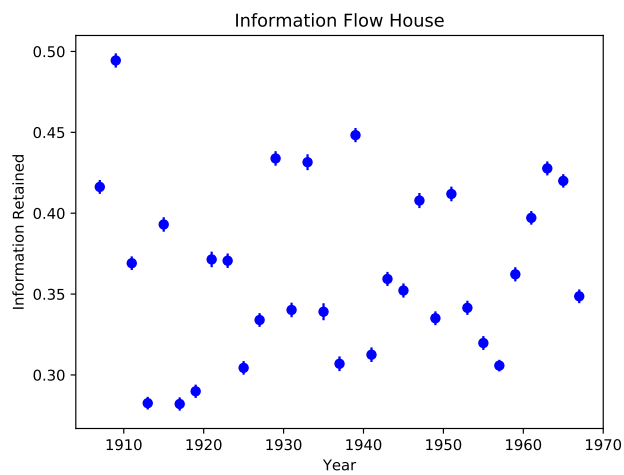
**Figure 3.16:** Flow in the dimension of the Senate roll call data for years 1907 to 1969 (60th to 90th Congress).



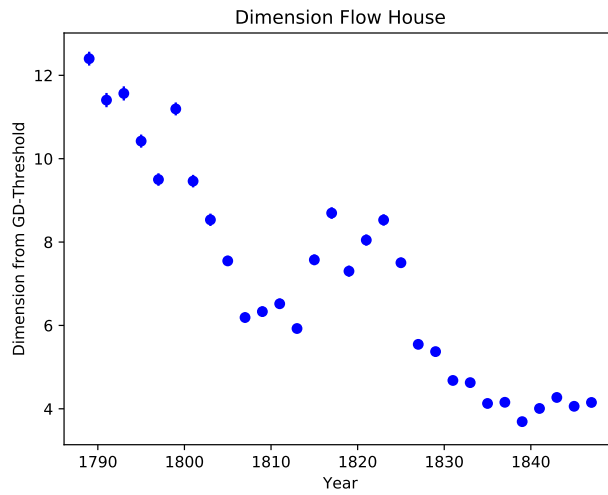
**Figure 3.17:** The flow in  $I(d = 1)(t)$  of the House roll call data for years 1789 to 1849 (1st to 30th Congress).



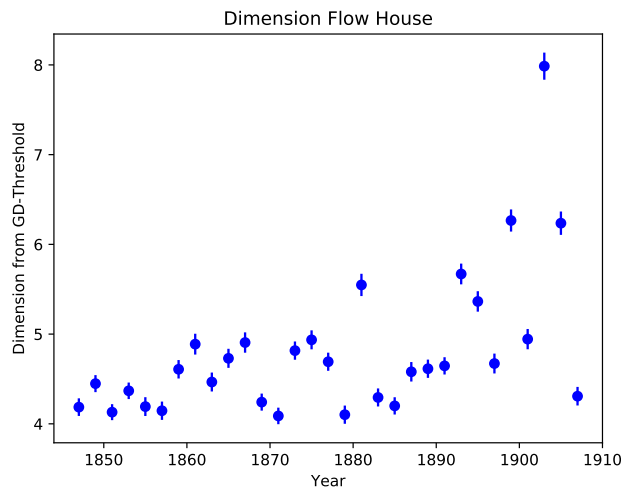
**Figure 3.18:** The flow in  $I(d = 1)(t)$  of the House roll call data for years 1847 to 1909 (30th to 60th Congress).



**Figure 3.19:** The flow in  $I(d = 1)(t)$  of the House roll call data for years 1907 to 1969 (60th to 90th Congress).

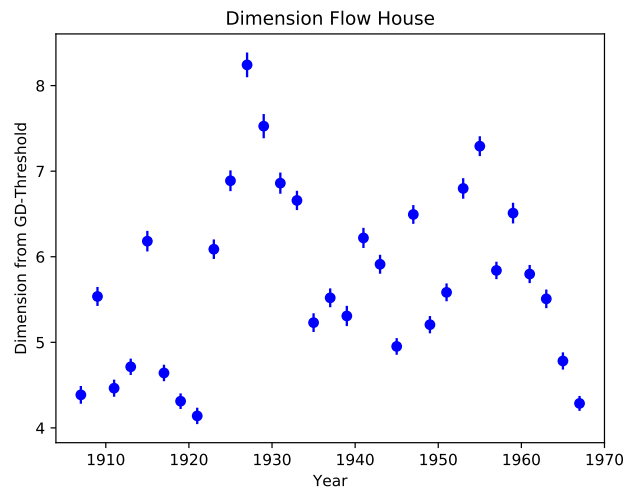


**Figure 3.20:** Flow in the dimension of the House roll call data for years 1789 to 1849 (1st to 30th Congress).



**Figure 3.21:** Flow in the dimension of the House roll call data for years 1847 to 1909 (30th to 60th Congress).





**Figure 3.22:** Flow in the dimension of the House roll call data for years 1907 to 1969 (60th to 90th Congress).

# Chapter 4

## Learned Persistent Homology

In this chapter, we will review *persistent homology* (also referred to as *persistence*), which is one of the main tools from TDA and generalizes hierarchical clustering methods to identify significant higher dimensional properties. It has been used to discover interesting and useful properties of data that had not been seen before in systems ranging from natural images [55] through the visual cortex [56] to RNA folding [59].

One of the motivations of using persistence is its robustness to data perturbations. See Section 2.4.5 for a broad overview of TDA motivations, [67] for robustness guarantees, and Figure 2.5 for a toy example of what this robustness means. In this chapter, we will investigate how noise in a dataset changes the topological information gathered from persistent homology. We do this by aggregating results gathered from running multiple trials, each of which outputs a set of Betti numbers. This set of results are then run through various classifiers in order to characterize the topology of the data. These concepts will be elaborated on in this chapter, and can also be reviewed in Section 2.4.3 and [47].

This chapter is meant to be an introduction to persistent homology, and we apply its tools to constructed noisy circles and noisy figure 8's. While we will see the utility of using persistence in being able to capture global information in these noisy datasets, we will also review some of

its issues. These issues will eventually lead us to modifying our approach in studying noisy data in Chapter 5, where we will construct a robust algorithm based on a topological method which does not share the persistence issues we will review. The main tool we will use from persistent homology is the *relative dominance* which will be defined in Section 4.1.2.

## 4.1 Background

### 4.1.1 Homology

In its broadest form, homology is a mathematical prescription that calculates algebraic properties of objects called chain complexes [47]. When these chain complexes consist of simplices, the homology that is calculated is a topological invariant of the space, and fundamentally relates to the number of  $n$ -dimensional holes of the space. It is thus a way to talk about isomorphisms of groups rather than homeomorphisms of spaces. This turns out to simplify the question of whether two spaces are fundamentally put together the same way or not. Formally, simplicial homology is defined as follows.

A *simplicial  $k$ -chain* ( $c_k$ ) is a sum of  *$k$ -simplices* ( $\sigma_k$ ):

$$c_k = \sum_i \alpha_i \sigma_k^i, \quad \alpha_i \in \mathbb{F} \quad (4.1)$$

where  $\mathbb{F}$  is some field. Each  $k$ -simplex can be thought of as a  $k$ -dimensional polytope. Thus, a 2-simplex represents a solid triangle; a 3-simplex represents a solid tetrahedron, etc. A collection of  $k$ -chains define a free Abelian group which is denoted as  $C_k$  - i.e.,  $c_k \in C_k$ . The *boundary operator*  $\partial_k : C_k \rightarrow C_{k-1}$ , is a linear homomorphism defined to act on  $\sigma_k = [v_0, v_1, \dots, v_k]$

$$\partial_k \sigma_k = \sum_i (-1)^i [v_0, v_1, \dots, \hat{v}_i, \dots, v_k] \in C_{k-1}$$

where “ $\hat{v}_i$ ” means this element is removed from the simplex. This definition leads to a flow of information in the various chain groups:

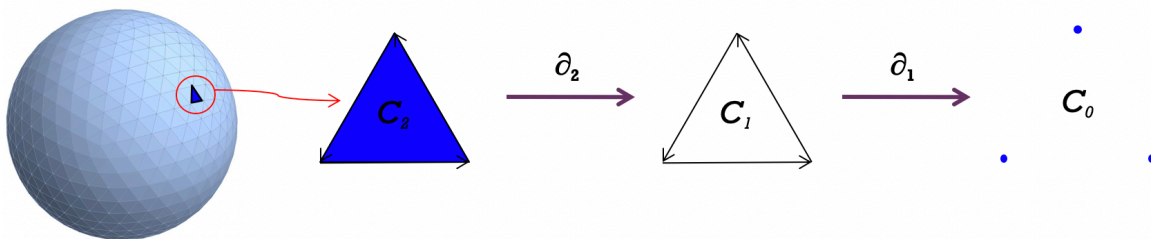
$$\dots \rightarrow C_{k+1} \xrightarrow{\partial_{k+1}} C_k \xrightarrow{\partial_k} C_{k-1} \rightarrow \dots$$

See Figure 4.1 for a depiction on what these simplices are and how the boundary operator acts on them.

Subgroups of this map can be defined that carry information about the global structure of the space. In particular, the *cycle group*  $Z_k \equiv \ker \partial_k$  and the *boundary group*  $B_k \equiv \text{im } \partial_{k+1}$ . Because  $\partial^2 \equiv 0$ , this implies  $B_k \subseteq Z_k \subseteq C_k$ . This condition is necessary so the homology group can be defined as the quotient group:

$$H_k \equiv Z_k / B_k = \ker \partial_k / \text{im } \partial_{k+1}$$

Each homology group,  $H_k$ , contains information about the existence of  $k$ -dimensional holes in the space. For instance, the torus has  $H_0 = \mathbb{Z}$ ,  $H_1 = \mathbb{Z}^2$ ,  $H_2 = \mathbb{Z}$  and all the remaining homology groups vanish. Refer to Hatcher’s text [47] for a full treatment of the subject.



**Figure 4.1:** Consider the 2-sphere as the topological space which can be seen as constructed from 2-simplices. The boundary operator acts on these simplices iteratively, at each step producing a set of lower dimensional simplices. Note, this figure leaves out the fact that  $\partial^2 \equiv 0$ .

## 4.1.2 Persistent Homology

The previous discussion on homology requires the spaces to be triangulable, that is able to be thought of as a sum of  $k$ -simplices. Topological data analysis allows us to extend this notion in order to triangulate datasets. Various methods of generalizing exist, each with their own distinct set of rules, that can be used to construct simplices from data. For each of these procedures, we choose the coefficients in Equation 4.1 to be in  $\mathbb{Z}_2$ .

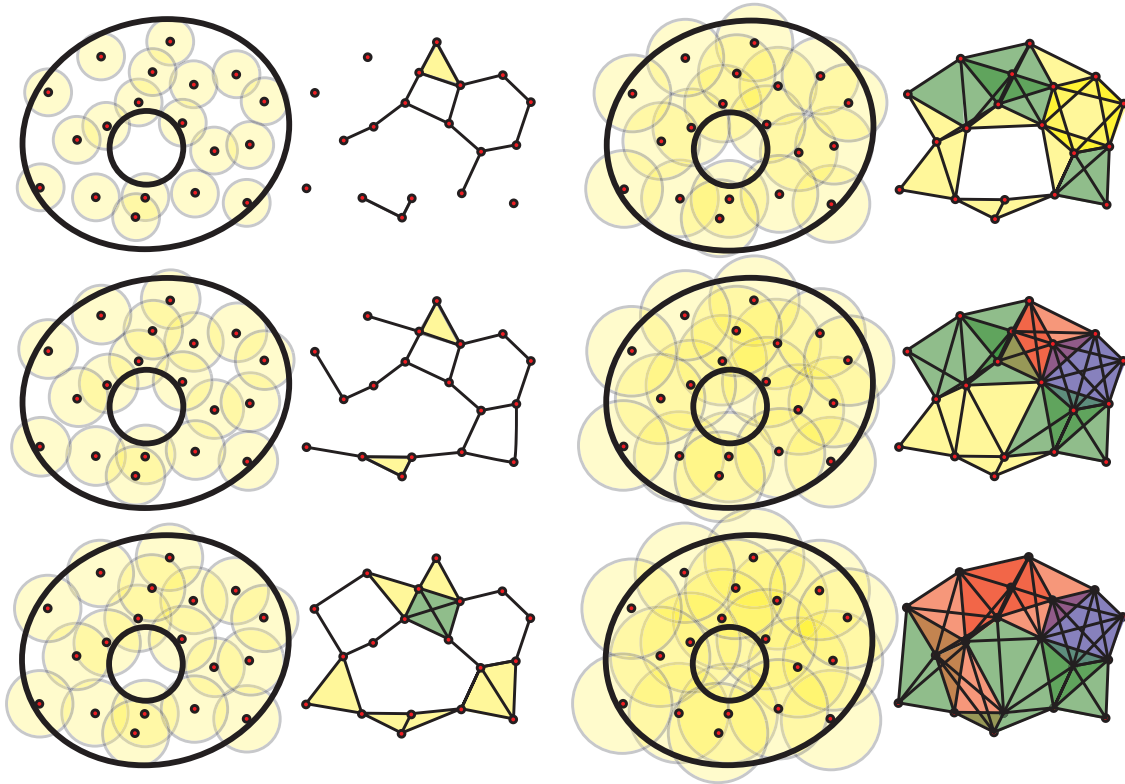
We use the terms *point cloud* and *data set* interchangeably. Let  $d(a, b)$  denote the distance in a metric space between points  $a$  and  $b$ . Let  $Z$  denote the point cloud. We refer to  $\epsilon$  as the *filtration value*, or simply the *filtration*. The vertex set consists of either the entire dataset's points or some subset of them. Refer to [48, 49, 50] for overviews of persistent homology. The remainder of this section will focus on the various ways in which we can construct the persistence.

### Vietoris-Rips Complex

Given a point cloud, the *Vietoris-Rips Complex* ( $R_\epsilon$ ) defines  $k$ -simplices as being determined by  $(k + 1)$ -tuples of points whose balls of radius  $\epsilon/2$  pairwise intersect [48] (see Figure 4.2 for a depiction of this procedure). The balls are drawn around each point in the point cloud, and the radius can be computed with an arbitrary metric. Specifically, to construct  $R(Z, \epsilon)$ :

1. The vertex set is  $Z$  or some subset of  $Z$
2. Edge  $[a, b]$  is in  $R(Z, \epsilon)$  iff  $d(a, b) \leq \epsilon$
3. Higher dimensional simplices are in  $R(Z, \epsilon)$  if all of its edges are in  $R(Z, \epsilon)$

One of the motivating reasons for this construction is that the union of the balls, which we interpret as being fundamentally representative of whatever topology the points came from, has a homotopy type that is closely related to the homotopy type of  $R(Z, \epsilon)$  (see [7]). Note that for a



**Figure 4.2:** A depiction of how the Rips complex changes as a function of  $\epsilon$ . 0-simplices are represented by the individual data points, 1-simplices by the black edges connecting them, 2-simplices by yellow, 3-simplices by green, 4-simplices by red, and 5-simplices by blue. Image from [48].

large enough filtration, the complex will become one connected component (see Figure 4.2). For a small enough filtration, there will be as many connected components as there are vertices.

The construction of  $R(Z, \epsilon)$  can be computationally expensive if the vertex set is large. Choosing the vertex set as a subset of  $Z$  reduces the computation necessary to construct the simplex set over the range of all filtration values.  $L \subset Z$  is called the *landmark set*, and points in it are chosen in one of two ways by selecting from  $Z$  [94]:

- *random* point selection: select points randomly from  $Z$ , the resulting set being  $L$
- *maxmin* point selection: first, choose a random point in  $Z$  to serve as the first point in  $L$ .

Each additional point in  $L$  is chosen from  $Z$  by maximizing  $d(z, l_i) \forall l_i \in L, z \in Z$

The size of  $L$  is variable depending on how large a vertex set is needed.

### Lazy Witness Complex

If a landmark set is taken as the vertex set for a Rips construction, information about the rest of the point cloud,  $Z \setminus L$ , is lost. To incorporate back in some of this information, we can construct the “Lazy Witness” complex  $LW(Z, L, \epsilon, \nu)$ :

1. The vertex set is  $L$
2. Edge  $[a,b]$  is in  $LW(Z, L, \epsilon, \nu)$  iff  $\exists z \in Z$  such that  $\max\{d(a, z), d(b, z)\} \leq D_\nu(z) + \epsilon$
3. All higher dimensional simplices are in  $LW(Z, L, \epsilon, \nu)$  if all of its edges are in  $LW(Z, L, \epsilon, \nu)$

$D_\nu(z)$  is defined to be the distance from  $z$  to its  $\nu$ th closest landmark point. A feature of the lazy witness complex is that it behaves like a Delauney triangulation of the space when  $\nu = 1$  ; for  $\nu = 0$ , the complex behaves similarly to  $R(Z, \epsilon)$  [94].

### Core Subsetting

Core subsetting is a procedure that helps uncover statistically significant topological structure in data. In a real world dataset, there is no reason a priori to expect the entire data set to have an interesting topological structure. Rather, we may expect subsets of the data to have the interesting structure. The procedure of core subsetting follows. First, start with an arbitrary  $n \times n$  metric space  $D$ , where  $d_{ij}$  represents the distance between the  $i$ -th and  $j$ -th data point:

$$D = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1n} \\ d_{21} & d_{22} & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{pmatrix}$$

The second step is to produce a *density* vector:

$$\Delta_k = \begin{pmatrix} \delta_1^k \\ \vdots \\ \delta_n^k \end{pmatrix}$$

Where  $\delta_i^k$  is defined to be the inverse of the distance from the  $i$ -th point in the metric space (i.e., the  $i$ -th row of  $D$ ) to the  $k$ -th closest neighbor. Hence,  $k$  is a parameter we can scan over. A large  $k$  can be thought of as giving a more global estimate of the topology; similarly, a small  $k$  gives a more local estimate. Additionally, there are other ways in which the density can be defined which we will not investigate here. The final step is to select a percentage “ $p$ ” of the densest points in  $\Delta_k$ . The densest points within the cutoff determined by  $p$  can then be used to form a smaller metric space:

$$\tilde{D} = \begin{pmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \cdots & d_{mm} \end{pmatrix}, m \leq n$$

## Analytical Tools

There are various quantities we examine when determining the fundamental topological structure of the point cloud and we compute these using javaPlex - a software built to construct persistent homology from an arbitrary point cloud [74]. The quantities we examine are:

- **Barcode Plots:** These depict the various generators of the different  $LW(Z, L, \epsilon, \nu)$  or  $R(Z, \epsilon)$ . The x-axis represents the filtration value; the y-axis represents, in no physically significant ordering, the different homology generators. A barcode exists for each  $H_n$ .



- **Betti Numbers:** Are the rank of the  $n$ -th homology group and are denoted as  $\beta_n$ . They are integers that count how many generators of a specific dimension exist, and for persistence, this value is dependent on the filtration value. For example, in persistence

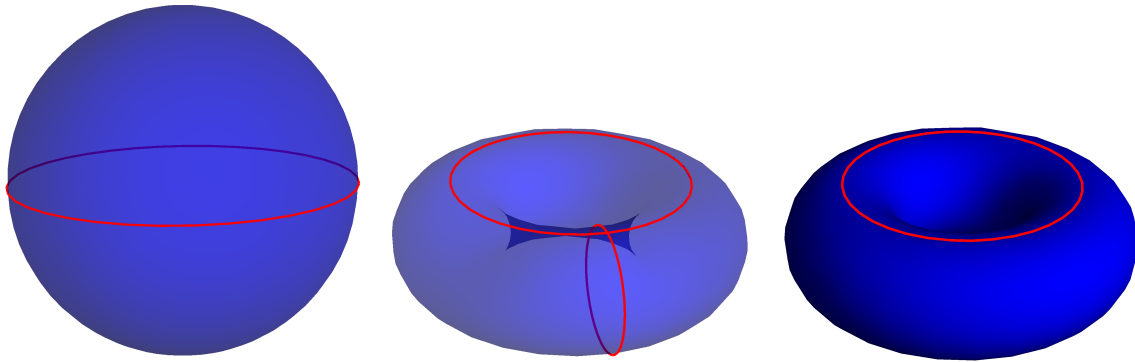
$$\beta_1 = |H_1(R(Z, \epsilon), \epsilon = 3)| = 2$$

means the first dimensional homology group for a Vietoris-Rips complex at a filtration of  $\epsilon = 3$  has Betti number equal to 2. In other words, it has 2 one-dimensional holes at this filtration, much as how the torus in Figure 4.3b has 2 one-dimensional holes. See Figure 4.3 for a depiction of Betti numbers for different topological spaces. Note that the list of Betti numbers can be merged to form a single vector that characterizes all the homological information about the space - we will use this concept later.

- **Relative Dominance [94]:** A few definitions are necessary in order to define the relative dominance:

1.  $\epsilon_0$  = The filtration value at which a topological structure appears, also referred to as the “birth” of the feature
2.  $\epsilon_1$  = The filtration value at which a topological structure disappears, also referred to as the “death” of the feature
3.  $\epsilon$  = The filtration value at which the complex becomes one connected component

*Relative Dominance* is defined as  $\delta_R = \frac{\epsilon_1 - \epsilon_0}{\epsilon}$ . Heuristically, we think of this as a meaningful quantity to look at because if  $\delta_R$  is large then the topological structure is likely significant, and on the other hand, a small  $\delta_R$  likely corresponds to the topological structure being noise. A priori it is not clear what is meant by a “small” or “large” relative dominance as this changes depending on the dataset and other choices made in the analysis. The idea then becomes, can we teach a computer to recognize what a large or small  $\delta_R$  is, given the



(a)  $S^2$ , the 2-sphere has Betti numbers:  $\beta_0 = 1, \beta_1 = 0, \beta_2 = 1$ . (b)  $T^2$ , the 2-torus has Betti numbers:  $\beta_0 = 1, \beta_1 = 2, \beta_2 = 1$ . (c) The solid 2-torus has Betti numbers:  $\beta_0 = 1, \beta_1 = 1, \beta_2 = 0$ .

**Figure 4.3:** Three spaces that are topologically distinct and characterized by their set of Betti numbers. The red lines represent potential 1-dimensional generators. In the case of  $S^2$  the generator can be contracted to a point, so  $\beta_1 = 0$ . The generators for the tori cannot be contracted, hence  $\beta_1 \neq 0$  for both.

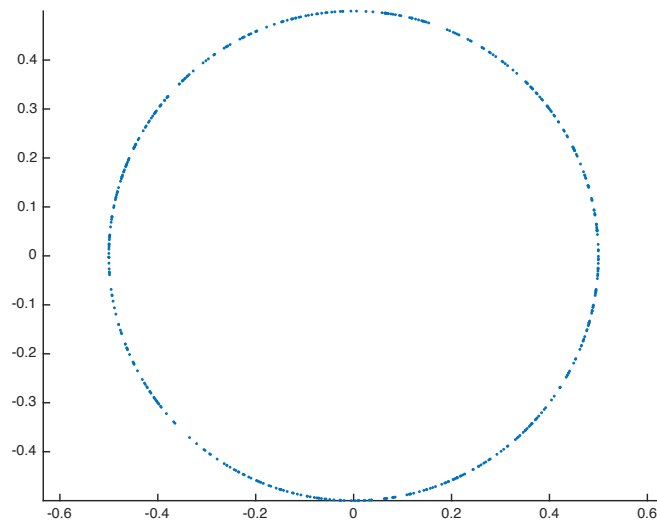
precise conditions of our project. In this chapter, we indeed show that this is possible in the case of simple noisy data, but we will also see some of the shortcomings to this approach, specifically of using  $\delta_R$ .

## 4.2 Ambiguities in Persistence

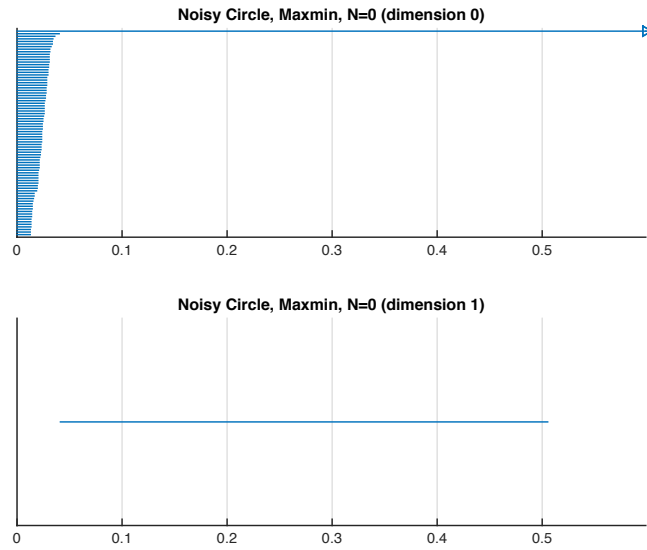
The tools to compute persistent homology we have briefly surveyed can work particularly well for “pure” and sufficiently sized datasets with no further analysis required to extract the true topological structure of the data. By “pure,” we ultimately mean that the topology is readily apparent by reading off the barcode, for instance in Figure 4.5. For many datasets though, reading off the topology from the barcode is not so straightforward, for instance in Figure 4.8. There are numerous reasons a dataset may produce uninterpretable (by human eyes) barcodes, a few of which include data that is: “fuzzy” (i.e., dispersed), large, noisy, or there may just be no interesting topology to begin with (i.e., the data can be a simple blob in  $n$ -dimensional space, an  $n$ -blob).

As an example of pure data, if we apply persistent homology to a dataset where we

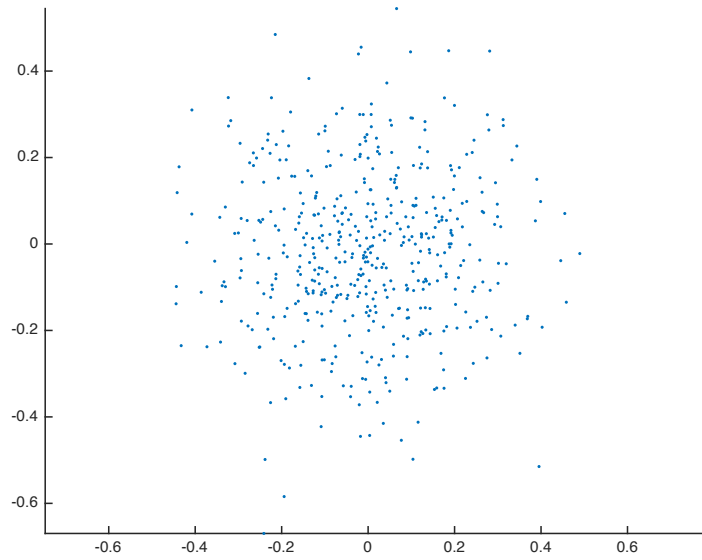
randomly sample a sufficient number of points from a circle, it is quite clear from the relative dominances that the structure is indeed circular. Figure 4.4 shows data randomly sampled from a circle and Figure 4.5 shows the barcode for this data. Notice that the barcode clearly shows the topological structure is a circle. As an example of fuzzy or noisy data, we take the pure circle from Figure 4.4 and add noise to each of the data points. The noise in this example consists of random samples from a 2D Gaussian (Figure 5.10). In Figure 4.7, the circle+noise point cloud still looks roughly circular, but looking at the barcode plot Figure 4.8, this is not as clear. In the noisy barcode, two issues make it hard to interpret: (1) there are numerous dominances each of which have comparable lengths, and (2) they are an order of magnitude shorter than for the circle without noise. By eye, there is some transition point where the barcodes become readable to non-readable, so we train a computer to extend this transition point.



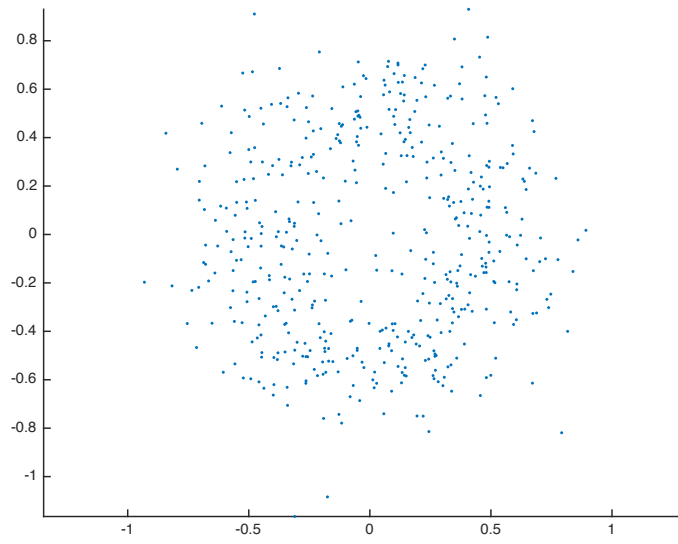
**Figure 4.4:** 500 randomly sampled points from a circle.



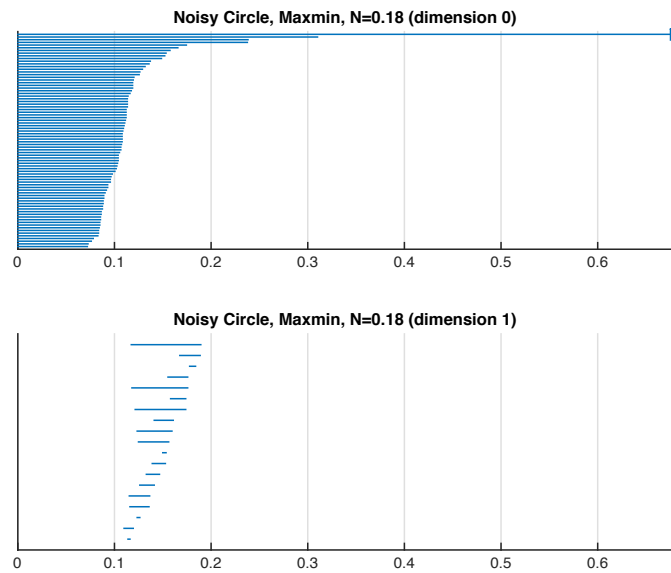
**Figure 4.5:** The barcode plot for Fig 4.4. For this example,  $\delta_R \approx 11.6$ . The top barcode represents how  $H_0$  changes with respect to filtration value; the bottom barcode represents the change in  $H_1$ . There is no significance to the ordering in either y-axis.



**Figure 4.6:** A 2D Gaussian distribution used as a noise source. The Gaussian has  $\sigma = 0.18$  for each axis.



**Figure 4.7:** A circle + Gaussian noise. The circle diameter is 1 and the Gaussian has  $\sigma = 0.18$  for each axis.



**Figure 4.8:** The barcode plot for Fig 4.7. For this example, the two largest relative dominances are:  $\delta_R \approx 0.23, 0.19$ . The top barcode represents how  $H_0$  changes with respect to filtration value; the bottom barcode represents the change in  $H_1$ . There is no significance to the ordering in either y-axis.

## 4.3 Data

The data we use to learn persistence consists of randomly sampled points from pure circles and figure-8's, with some degree of noise added to each of these points. The noise we add serves to make the data closer to what a real world data set may consist of, and consists of data randomly sampled from a Gaussian in  $n$ -dimensional space. If we call  $\mathbb{G}$  an  $n$ -dimensional Gaussian,  $\mathbb{X}$  the pure data space, and  $\sigma$  the symmetric variance of the Gaussian, then the space  $\tilde{\mathbb{X}}$  we study consists of points whose coordinates are given by:

$$\begin{aligned}\tilde{x}_i &= x_i + g_i(\sigma), \\ \tilde{x} &\in \tilde{\mathbb{X}}, x \in \mathbb{X}, g \in \mathbb{G}\end{aligned}$$

The range of the pure data is typically around one (more on this in the next section). Given the range of our data, when  $\sigma \equiv 0$  the data consists of the pure data; when  $\sigma \sim 0.5$  the magnitude of the noise is on the order of the magnitude of the data, in which case the data will be too noisy to uncover any topological structure. In fact, we expect the data to be too noisy except for relatively small values of  $\sigma$ . In constructing the data in conjunction with persistence, we choose various parameters which we summarize below and in Table 4.1:

1.  $N \equiv$  number of generated points in  $\mathbb{X}$ .
2.  $N_l \equiv$  number of landmark points from  $\tilde{\mathbb{X}}$  (see Section 4.1.2). We call the set of these points  $\tilde{\mathbb{X}}_l$  (i.e.,  $\tilde{\mathbb{X}}_l \subset \tilde{\mathbb{X}}$ ). The complexity of computing the homology of a dataset scales with this parameter. We set  $|N_l| = 80$  based on heuristic observations in which we compare the computation time versus the quality of the barcode.
3.  $\sigma \equiv$  variance of the Gaussian (i.e., a noise parameter). We vary this from  $0.10 \rightarrow 0.50$  in a step size of 0.05.

4.  $k \equiv$  core subset parameter, which we set to 1 in this analysis. Zigzag persistence can be used to investigate the topology when this parameter changes [95]. Again, this is set heuristically.
5. We choose a Lazy complex with  $\nu = 0$ , maxmin selection, and Euclidean metric

**Table 4.1:** Parameters of Generated Data

<b>Structure</b>	<b>Landmark Selection</b>	<b>Diameter</b>	$N$	$p$	$k$	$N_l$
circle	maxmin	1	500	50	1	80
figure-8	maxmin	1	1000	50	1	80

Note that increasing  $N$  or  $N_l$  may bias the analysis towards better results (i.e., the relative dominances show more clearly the underlying topological structure). Up to a certain point, increasing  $N$  more clearly defines the pure data structure we analyze and increasing  $N_l$  gives us more information about this total space. We heuristically set  $N_l$  based on observing how the relative dominance changes when it is varied.

The purpose of this analysis is to present a procedure which quantifies the significance of  $\delta_R$  in the barcodes, and in doing so we should be able to determine its topology. We present both supervised and unsupervised approaches to learning this structure in Sections 4.4.1 and 4.4.2, respectively.

Since we are working with two dimensional point clouds (the circle and figure-8), we only look up to  $H_1$ . To build up the mathematical structure of  $\delta_R$ , we submit 300 runs for a set of given parameters. Each run produces a barcode plot with the same set of parameters as the previous run, the only difference between them is the inherent randomness involved which may produce different results. The randomness comes from producing the pure point cloud  $\mathbb{X}$  each run, from the landmark selection  $\tilde{\mathbb{X}}_l$ , and from producing the Gaussian noise  $\mathbb{G}$  for each run. To give an example of this for the noisy circle, we produce 300 runs following the procedure below:

### One Run

- Produce 500 points randomly sampled from a circle with diameter=1
- Set  $\sigma$  and add Gaussian noise to each coordinate in the data
- Core subset the data, with  $p = 50\%$  and  $k = 1$
- Choose maxmin points from the core, with  $N_l = 80$
- Produce barcode then relative dominances

## 4.4 Learning Persistence

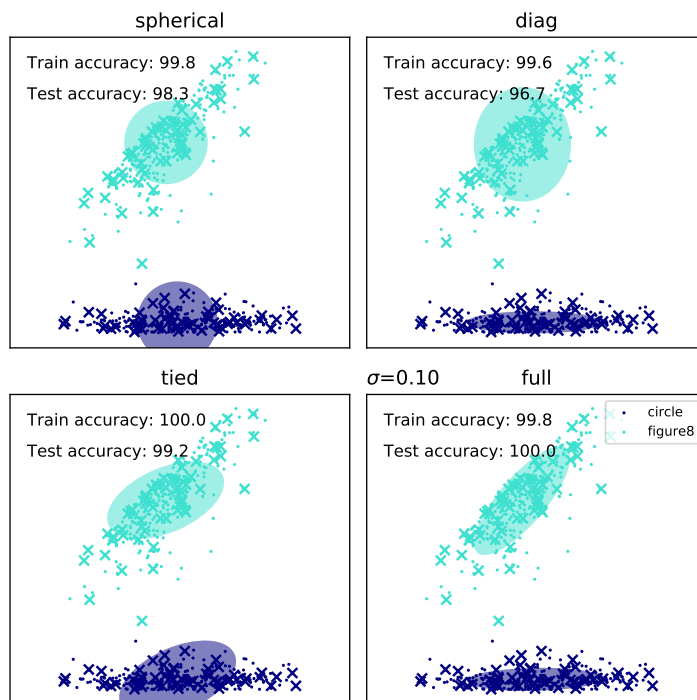
In the following sections, we present supervised and unsupervised approaches (Section 2.1) in analyzing the structure of  $\delta_R$  from the barcodes. The supervised approaches we use include Gaussian mixture models (“GMM”) [36], random forests (“RF”) [44] and neural networks (“NN”) [6]. In the unsupervised approach, we will use a Kolmogorov-Smirnov (“KS”) test [96, 97]. In the supervised cases, we will train up accurate classifiers and in the unsupervised method we will ultimately look at  $p$ -values with respect to noise.

### 4.4.1 Supervised Learning of Persistence

We will use three classifiers in our supervised approach: GMMs, RFs, and NNs. Gaussian mixture models are generative, while random forests and the neural network we use are discriminative (see Section 2.4.4). At first thought, a GMM would seem to be a better classifier since we might expect the Gaussian noise, which is added to the pure data, to permeate through to the structure of  $\delta_R$ . While a GMM model still maintains relatively good classification, the random forest and neural network outperform it, especially for larger noise values. In Figures 4.15 and 4.16 we see the distributions of  $\delta_R$  in 1-dimensional slices are indeed not Gaussian. The data we



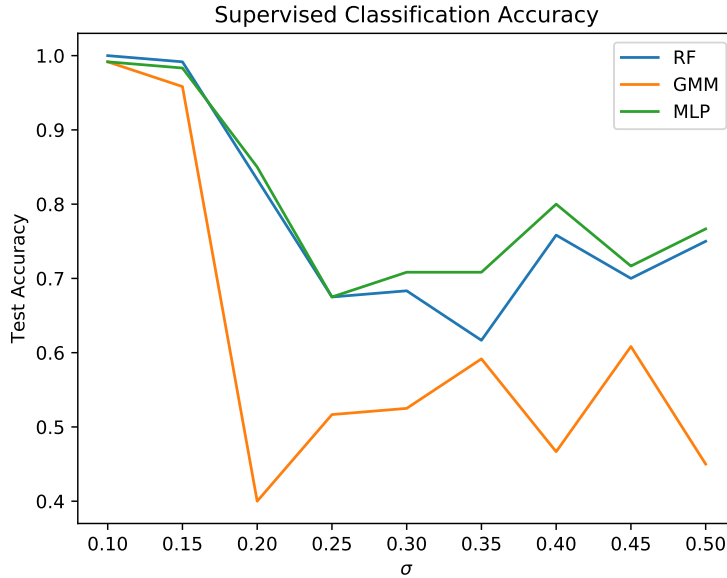
use as input to these classifiers is another point cloud in  $\mathbb{R}^5$  where the coordinates correspond to the largest  $H_1$  relative dominances for a given run, arranged in descending magnitude. For instance, the first coordinate corresponds to the largest relative dominance and the last coordinate corresponds to the smallest. If a relative dominance doesn't exist for one of the coordinates, we populate that coordinate with a zero.



**Figure 4.9:** A GMM with four different covariance matrices (see Appendix C) applied to classifying noisy circles and figure-8's with  $\sigma = 0.1$ . Depicted here is a projection of the GMM and first two relative dominances to  $\mathbb{R}^2$ .

A GMM consists of fitting the data to a weighted sum of Gaussians:  $p(x; \theta) = \sum_{c=1}^C \alpha_c \mathcal{N}(x; \mu_c, \Sigma_c)$ , where  $\alpha_c$  is the weight of the component  $c$ ,  $0 < \alpha_c < 1$ ,  $\sum_{c=1}^C \alpha_c = 1$  and  $\theta$  is a list of parameters. See Appendix C for a further description of a GMM and its parameters.

In Figure 4.10 we see that the random forest and neural network retain high accuracies even for high levels of noise, which means they are able to detect the mathematical structure of  $\delta_R$  and generalize to unseen instances. Even at  $\sigma = 0.5$ , these classifiers are able to predict the



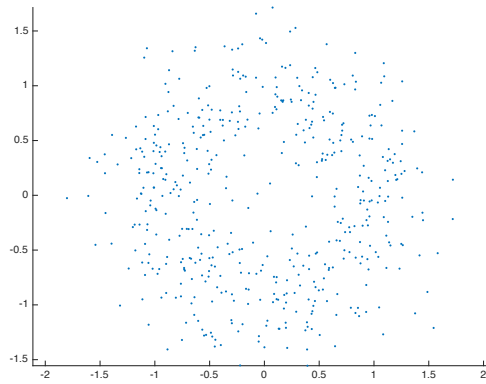
**Figure 4.10:** Test accuracy as a function of  $\sigma$  for a GMM, RF, and NN. Even at  $\sigma = 0.5$ , the RF and NN are able to predict the topology with 75% accuracy on test instances. The NN we use is a shallow multilayer perceptron (“MLP”).

topology with 75% accuracy on test instances. In Figures 4.11 and 4.12 we see how large noise actually affects the datasets.

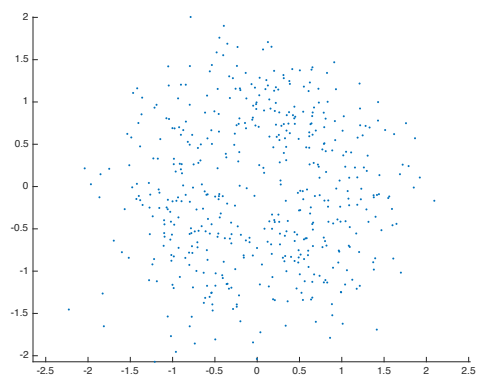
## 4.4.2 Unsupervised Learning of Persistence

In this section, we present an unsupervised approach in learning the persistence. The idea here is to aggregate the relative dominances for the various  $H_k$  homology groups into histograms, and to use just these histograms to determine whether a generator in each of the  $H_k$  is statistically significant. More specifically, the approach will be to populate  $n_k$  histograms for each of the  $H_k$  (in our case, we only analyze  $H_1$  and set  $n_1 = 3$  for simplicity) for each dataset where the statistics are generated by the runs as in Section 4.3. The value of  $n_k$  sets the limit on the number of  $k$ -dimensional generators we will analyze to see if they’re statistically significant. If we expect  $m$  significant generators for  $H_1$ , for instance,  $n_1$  should be set higher than  $m$ .

The histograms correspond to the three largest relative dominances (i.e.,  $n_1 = 3$ ), param-

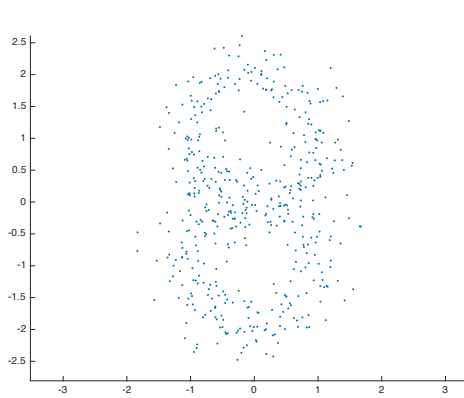


(a) Noisy circle with  $\sigma = 0.3$

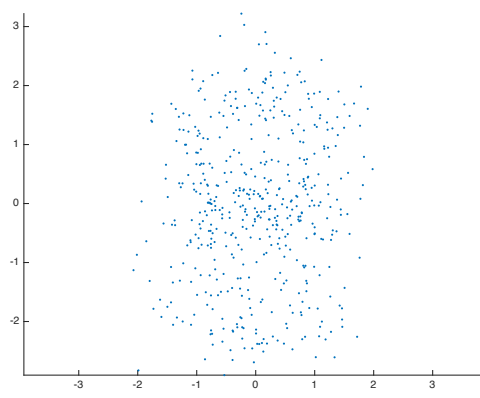


(b) Noisy circle with  $\sigma = 0.5$

**Figure 4.11:** Noisy circles for two values of  $\sigma$ . At  $\sigma = 0.3$  accuracy  $\sim 70\%$  and at  $\sigma = 0.5$  accuracy  $\sim 75\%$  for both the RF and NN.



(a) Noisy figure8 with  $\sigma = 0.3$



(b) Noisy figure8 with  $\sigma = 0.5$

**Figure 4.12:** Noisy figure-8's for two values of  $\sigma$ . At  $\sigma = 0.3$  accuracy  $\sim 70\%$  and at  $\sigma = 0.5$  accuracy  $\sim 75\%$  for both the RF and NN.

eterized by the amount of noise added to the data. We expect the distribution to be positively skewed, i.e., with a longer right tail, as the predominance of fleeting or small  $\delta_R$  will likely be more numerous in noisy data. Below, we show the histograms in Figures 4.15 and 4.16.

For small values of noise, it is clear what the homology is for both the circle and figure-8 from these histograms. However, even for these small values of noise there is still a large standard deviation, and hence a single sampling of a relatively clean dataset in general may be insufficient to determine the homology (which is related to other issues brought up in Section 4.5). As noise is increased, the average of the relative dominances shifts left, and the standard deviation decreases.

Considering the distributions of the circle (Figure 4.15) we see that the first histogram for low noise is clearly shifted right with respect to all higher order distributions. Thus, an unsupervised approach in determining the homology would be to test which lower order histograms are sufficiently different from higher order ones. A two sample Kolmogorov-Smirnov (“KS”) test can be used to determine how distinct any pairs of the histograms are [96, 97]. The KS test returns a  $p$ -value which is the probability that, if the null hypothesis were true (being that the two distributions were drawn from the same model), we would observe the two distributions we constructed. <sup>1</sup> If we set a threshold on this value, say  $p = 5\%$ , then when the KS test returns a value  $\leq 5\%$ , we will say that the distributions are different.

If we use  $\delta_R^{(i)}$  to represent the distribution of the  $i$ -th largest relative dominance (i.e.,  $\delta_R^{(1)}$  is the largest), we would like to see the following  $p$ -values:

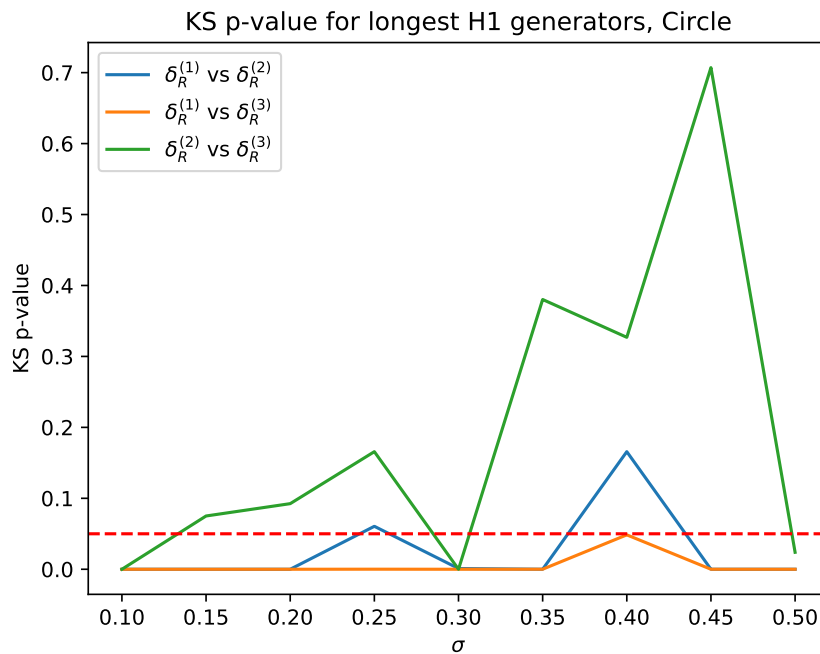
**Table 4.2:** The  $p$ -values we would like to see for the circle and figure-8 for the three comparisons in  $\delta_R^{(i)}$ . A  $p$ -value  $\leq 5\%$  means the distributions are different; a  $p$ -value  $> 5\%$  means the distributions can be considered the same.

Data	$\delta_R^{(1)}$ vs $\delta_R^{(2)}$	$\delta_R^{(1)}$ vs $\delta_R^{(3)}$	$\delta_R^{(2)}$ vs $\delta_R^{(3)}$
circle	$\leq 5\%$	$\leq 5\%$	$> 5\%$
figure-8	$> 5\%$	$\leq 5\%$	$\leq 5\%$

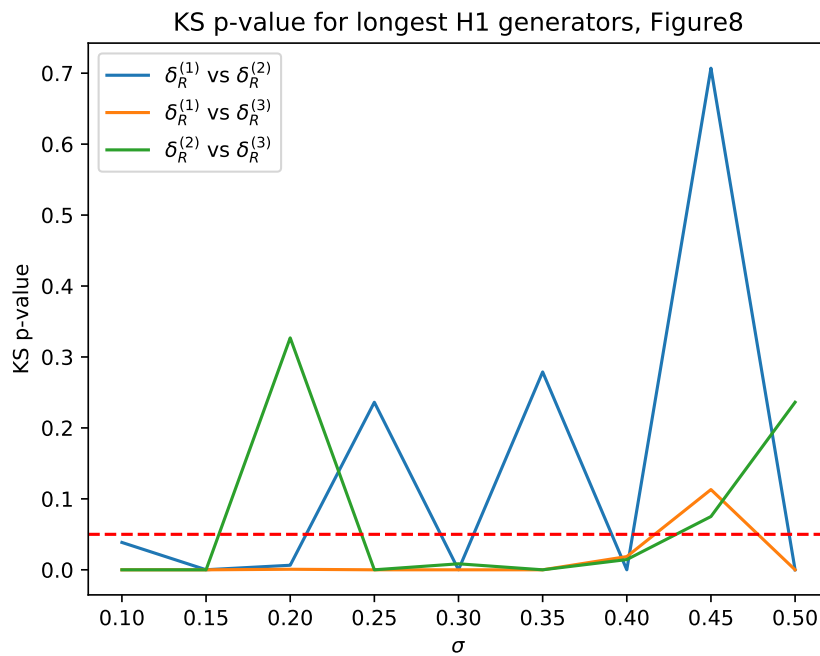
<sup>1</sup>Technically speaking, the  $p$ -value tells us the probability of observing a  $D$ -value, a measure of statistical difference in KS testing, as discrepant or more so than what we observed.

These cutoffs are determined by the assumption that noisy  $\delta_R$  distributions are fundamentally different from topologically significant  $\delta_R$ . In a future analysis, these cutoffs or different criteria may be more appropriate.

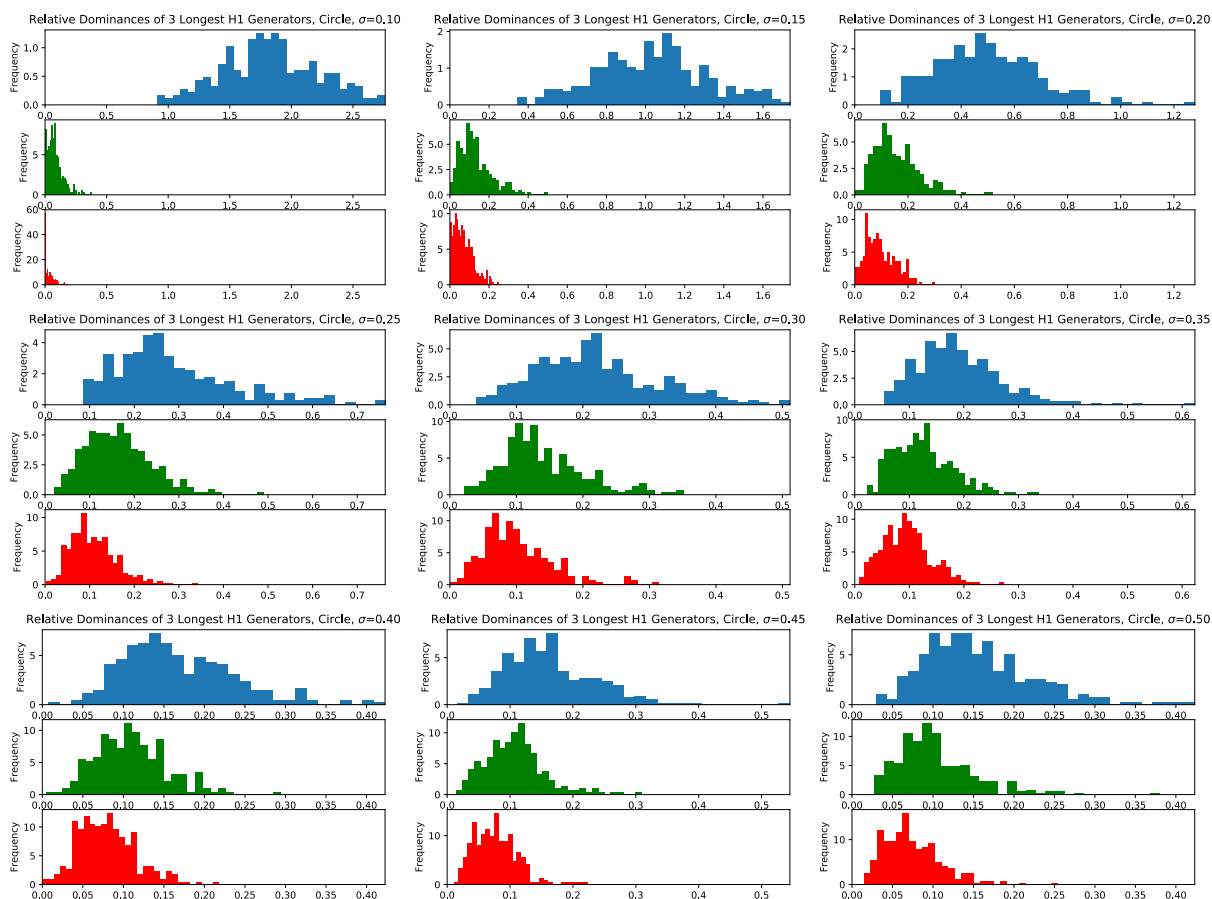
Figures 4.13 and 4.14 show the  $p$ -value between these pairwise comparisons with respect to noise level. We see that the distributions do fairly well in maintaining the cutoff requirements from Table 4.2, but two improvements can be made here: (1) reduce the overall number of violations to Table 4.2 and (2) smooth out the jumpiness in these plots. Preprocessing the  $\delta_R^{(i)}$  distributions improves this behavior to an extent, and indeed Figures 4.13 and 4.14 were computed on  $\delta_R^{(i)}$  that had been scaled to the interval  $[-1, 1]$ . In a future analysis, modifying the cutoffs, the preprocessing, or statistical test may improve the quality of these plots.



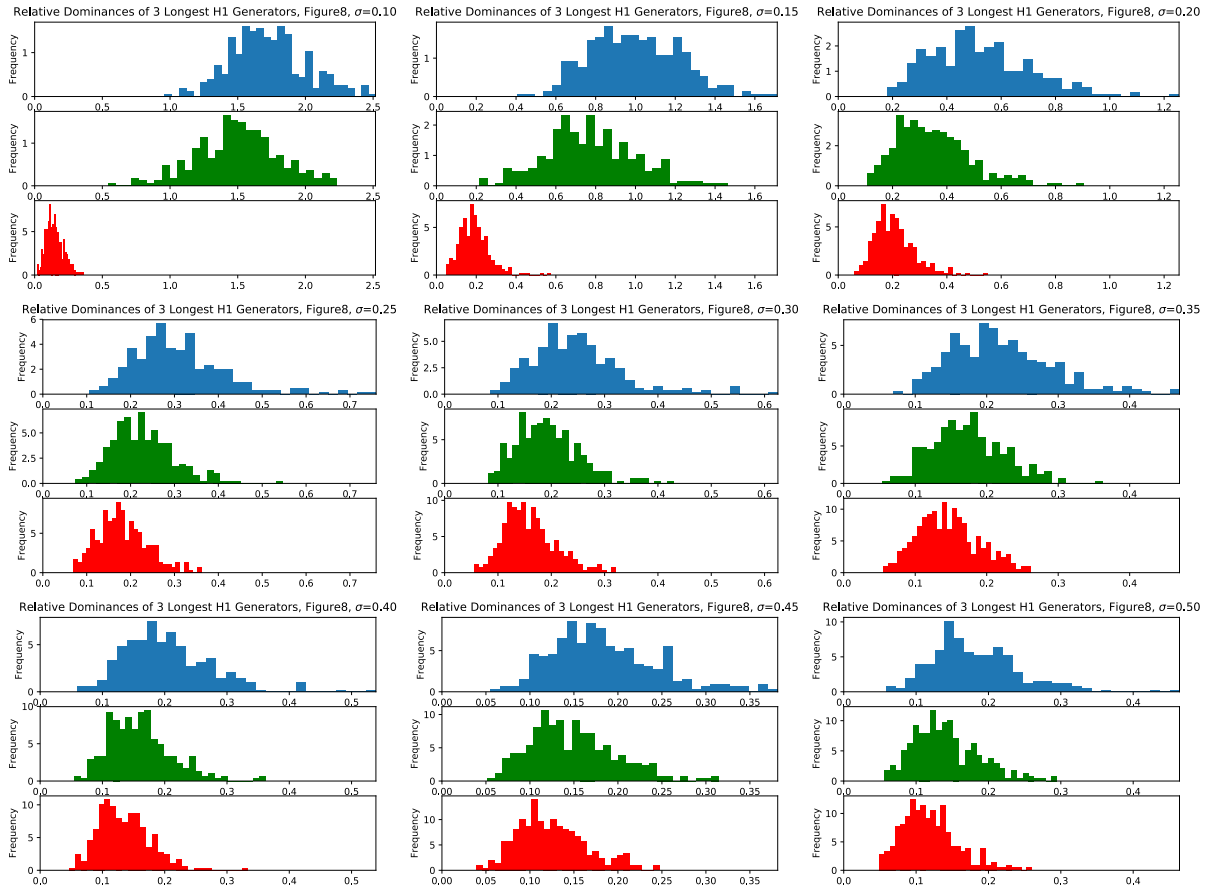
**Figure 4.13:**  $p$ -value with respect to noise for pairwise comparisons of  $\delta_R^{(i)}$  for the circle. The red line determines the 5% cutoff. We would like to see the following conditions met:  $\leq 5\%$ ,  $\leq 5\%$ ,  $> 5\%$ .



**Figure 4.14:**  $p$ -value with respect to noise for pairwise comparisons of  $\delta_R^{(i)}$  for the figure-8. The red line determines the 5% cutoff. We would like to see the following conditions met:  $> 5\%$ ,  $\leq 5\%$ ,  $\leq 5\%$ . Modifying the cutoffs, the preprocessing, or statistical test may improve the quality of this plot and smooth out its jumpiness.



**Figure 4.15:** Histograms for the 3 largest relative dominances of the circle. At each noise level, blue is  $\delta_R^{(1)}$ , green is  $\delta_R^{(2)}$ , red is  $\delta_R^{(3)}$ . These colors do not correspond to the  $p$ -value measurements.



**Figure 4.16:** Histograms for the 3 largest relative dominances of the figure 8. At each noise level, blue is  $\delta_R^{(1)}$ , green is  $\delta_R^{(2)}$ , red is  $\delta_R^{(3)}$ . These colors do not correspond to the  $p$ -value measurements.



## 4.5 Ambiguities in Persistence: Part Deux!

The supervised classifiers in Section 4.4.1 were able to maintain a high level of test accuracy even for large noise values. This is somewhat surprising at first thought given that Figures 4.11b and 4.12b appear to have little topological structure. The core subsetting could be one of the reasons why the overall algorithms are still able to pick out the topological structure, or perhaps it's just the contrived nature of the data itself. Speaking to the latter point, all the data were produced under the same set of assumptions between the runs (i.e., same diameters, same density, etc.). However, if we were to feed in a figure-8 with diameter=2 through the entire classification algorithm, it would likely be confused due to issues in  $\delta_R$ . Some of these issues come from the fact that, though it is a quantity derived from topological notions, it is not a completely topological quantity.

For instance, the relative dominance is not scale invariant, which is indeed one of the motivations of using TDA in the first place (Section 2.4.5). To give an example, consider two pure circles with no noise each with the same density of points, however one has two times the radius of the other. If circle A is the smaller of the two, for  $H_1$  we would expect to find:

$$\begin{aligned}\delta_R^A &= \frac{\epsilon_1 - \epsilon_0}{\epsilon} \\ \delta_R^B &= \frac{\epsilon'_1 - \epsilon_0}{\epsilon} = \frac{2(\epsilon_1 - \epsilon_0)}{\epsilon} = 2\delta_R^A\end{aligned}\tag{4.2}$$



Note that the birth time is the same in either case since the densities of each circle are the same. However, the *(death) – (birth)* time is 2 times longer in the case of circle B as it has 2 times the radius of circle A. Thus,  $\delta_R$  can change drastically between datasets even if they are topologically equivalent.

A second issue with using persistence to study the topology of an object is that it is constructed to capture solely the  $n$ -dimensional hole information in the data. If this is all that

is required for understanding the data, then persistence can be a useful tool. However, in many cases, there is interesting structure to the data that can be missed by  $\delta_R$ . For instance, take the letter “A” and the letter “O” as the datasets in question. Theoretically, persistence should say both these objects are topologically equivalent to a circle. While this is true from the homological standpoint, it certainly coarse grains the dataset too much if we are interested in distinguishing between these objects. Now, given the considerations in previous sections, namely that barcode plots can look much different even for two different (noisy) circles, there could be the hope that the barcode plot structure actually does resolve the “A” vs “O” structure. But if we start to shorten the legs of the “A”, at some point persistence will certainly miss this structure.

An alternate TDA method to persistence utilizes objects called *mappers*, which we will review in Chapter 5, that can capture the information about the legs of “A”. See Table 4.3 for a summary of how persistence and mapper should theoretically resolve structure in “A” and “O”.

**Table 4.3:** The data  $\mathbb{X}$  is either the letter “A” or “O”. Theoretically, persistent homology will return a 1 for the first two Betti numbers; mapper will return a graph that looks different between the datasets.

$\mathbb{X}$	<b>PH</b> ( $\beta_0, \beta_1$ )	<b>Mapper</b>
The letter “A”	(1, 1)	
The letter “O”	(1, 1)	

Due to these issues in  $\delta_R$ , we will use mapper objects in Chapter 5 as a basis for a robust image recognition algorithm.

## 4.6 Acknowledgements

We would like to thank the developers of javaPlex, and specifically to Henry Adams for all the insight he provided into the world of persistent homology.

# Chapter 5

## Learned Mappers

Topological data analysis aims to extract topological quantities from data, which tend to focus on the broader global structure of the data rather than local information. The Mapper method, specifically, generalizes clustering methods to identify significant global mathematical structures, which are out of reach of many other approaches. We propose a classifier based on applying the Mapper algorithm to data projected onto a latent space. We obtain the latent space by using PCA or autoencoders. Notably, a classifier based on the Mapper method is immune to any gradient based attack, and improves robustness over traditional CNNs. We report theoretical justification and some numerical experiments that confirm our claims.

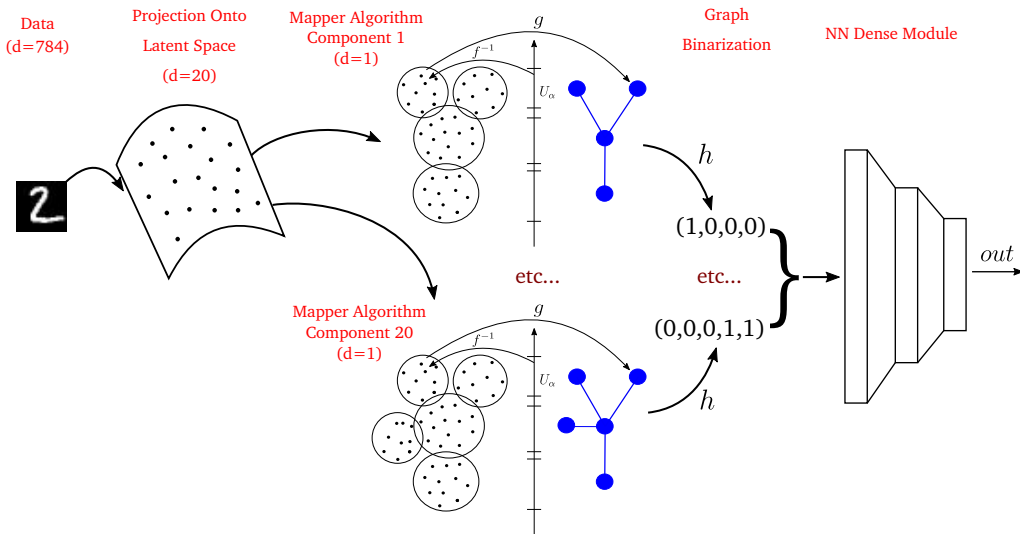
### 5.1 Introduction

Deep neural networks [98, 6] are well known to be not robust with respect to input image perturbations, and to be susceptible to so-called “adversarial attacks” which are designed by adding to images perturbations that are typically non-perceptible by humans [99, 100, 101]. In this chapter we explore opportunities for combining deep learning techniques with a well known *topological data analysis* (TDA) algorithm – *the Mapper* [51], which we use to create classifiers with improved robustness. First, the training data is projected onto a latent space.

The latent space in the simplest variant is constructed using *PCA* components, and we also use nonlinear projections by utilizing various *autoencoders* [98, 102, 103, 104]. Then, a discrete graph representation (Mapper) is assigned to the training data projected onto the *latent space*. Having this trained graph structure built, any input can be binarized, by assigning the binary vector representing the nodes in the graph to which it belongs. We emphasize that such discretization step makes our algorithm essentially immune to any white-box *gradient based adversarial attack*. The test data is treated by a special mapping procedure that is essentially performing a weighted *k*-nearest neighbor search in the preimage of some portion of the latent space in order to compute a vectorized representation of the testing points. We apply the algorithm we have developed, using methods from topological data analysis and more traditional approaches, to the *MNIST* [12] datasets as an application of robust computer vision.

The main ingredient in this algorithm is our implementation of a topological object, called a *Mapper*, which captures global information about the data space. Intuitively, these objects allow for some “stretchiness” in the data while still producing the same desired output [66]. In general, methods in topological data analysis are robust to perturbations in data because the overall global structure remains relatively unchanged, and these methods capture this information. Owing to this property, the variance in our predictive model is reduced, and due to an ensemble approach we implement the bias is additionally reduced. Hence, our algorithm resolves, to an extent, the *bias-variance tradeoff*. We will note that Mappers have been used to classify error modes for CNNs when applied to MNIST data [68].

The original software to produce Mapper objects is *Python Mapper* [52]. The code we have constructed for this analysis is a prototype implementation and is built around an R implementation of the Python Mapper software [72]. Using Mapper as a means to compare shapes has been done before in [51] and [105]. Our approach is different in that we utilize traditional machine learning approaches in conjunction with Mapper. We additionally apply our method to more extensive data and construct a measure of robustness of our approach. Our algorithm is



**Figure 5.1:** Illustration of our MC method. In practice we use not just one Mapper object but a whole committee of mappers (see Section 5.2).

general and can likely serve as a robust classifier for data other than image data. For the purpose of this chapter, we interpret inputs as  $N_{\text{in}}$  dimensional real-valued vectors, and denote the full space of inputs by  $\mathcal{X} \subset \mathbb{R}^{N_{\text{in}}}$ . We denote a set of examples in some given benchmark dataset by  $\bar{\mathcal{X}} \subset \mathcal{X}$ , which is split into the training and the testing sets, i.e.,  $\bar{\mathcal{X}} = X_{\text{train}} \cup X_{\text{test}}$ . We use the notation  $X$  to denote a general dataset, i.e., either  $X_{\text{train}}$  or  $X_{\text{test}}$  when the distinction between these is not important. The notation for the space of images  $\mathcal{X}$  will become useful later, especially when we discuss perturbed data.

### 5.1.1 What is Mapper?

The Mapper method [51] is a discretized analog of *Reeb graphs* [7, 106], which are tools used in *Morse theory* [107]. Both Mapper and Reeb graphs provide topological information pertaining to connectivity of the space. More precisely, they describe changes in level sets in a space given a function over this space. Some motivations for using these approaches to understand data consist of: the ability to get a higher-level understanding of the structure of data

by determining clustering information (which is based on clusters in  $X$  and how various functions behave on  $X$ ), and the low computation cost of producing these topological networks. They have been used for a number of different applications, including: the discovery of significant clusters in breast cancer gene expression, the classification of player performance in the NBA, and voting behavior in the United States Congress [60]; to study RNA hairpins to identify dominant folding paths [59].

We denote a specific Mapper object as  $M = M(X, f)$  and the space of Mapper objects by  $\mathcal{M}$ . Below, we describe the Mapper algorithm and provide specific examples of Mapper objects.

### 5.1.2 Mapper Algorithm

#### INPUT:

- The dataset  $X_{\text{train}} \subset \mathbb{R}^{N_{\text{in}}}$ .
- The choice of metric for computation of the pairwise distances (fixed to Euclidean).
- The function  $f: \mathcal{X} \rightarrow \mathbb{R}$ , referred to as the “lens” or “filter function.”<sup>1</sup>
- The number of intervals in the open cover  $\equiv n_{\text{int}}$  of  $\mathbb{R}$ , the percent overlap of the intervals (also referred to as the *gain*), the number of bins in a histogram consisting of distances at which clusters merge in a single-linkage clustering  $\equiv n_{\text{bins}}$ . We set  $n_{\text{bins}} = n_{\text{int}} = 10$  and  $\text{gain} = 0.33$  (more on this in Section 5.3).

#### OUTPUT:

- The Mapper object  $M \in \mathcal{M}$  (undirected graph encoding the clustering of data and the intersection structure (see Remark 5.1)).

#### begin

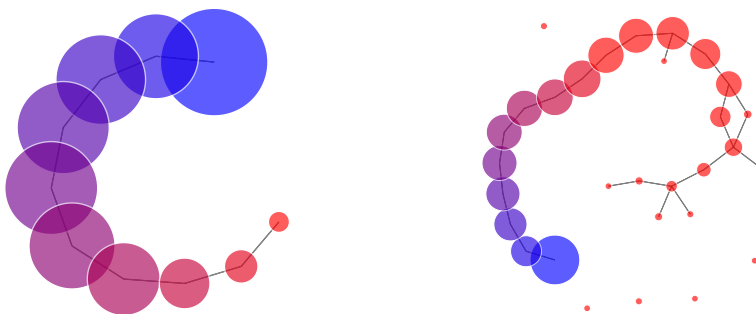
1. Set  $I = \text{im}(f)$ . Choose an open cover  $\{U_\alpha\}$  for  $I$ .

---

<sup>1</sup>The Mapper algorithm does not require a mapping to one dimension, and indeed it is possible to replace this step with  $f: \mathcal{X} \rightarrow \mathbb{R}^m$  for arbitrary  $m$ .

2. Set  $V_\alpha = f^{-1}(U_\alpha)$ . Then  $\{V_\alpha\}$  is an open cover for  $X$ .
3. Refine  $\{V_\alpha\}$  to  $\{V_{\alpha,i_\alpha}\}$  where  $i_\alpha$  indexes the  $N_\alpha$  components of  $X_{\text{train}} \cap V_\alpha$  defined by connecting points that have distance less than  $\epsilon > 0$ , where  $\epsilon$  is dependent on  $n_{\text{bins}}$ . Let  $\tilde{N} = \sum_\alpha N_\alpha$ .
4. Let  $\{(\alpha, i_\alpha)\}$  label the  $\tilde{N}$  vertices of a simplicial complex. It is useful to think of data points living in these vertices.
5. Connect vertices labeled  $(\alpha, i_\alpha)$  and  $(\alpha', i_{\alpha'})$  iff  $V_{\alpha,i_\alpha} \cap V_{\alpha',i_{\alpha'}} \neq \emptyset$ .

**end**

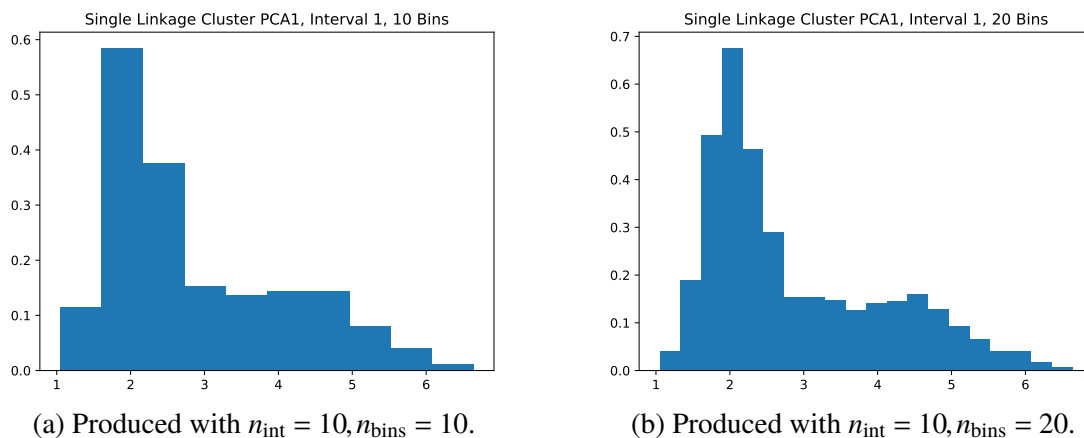


**Figure 5.2:** Mapper objects computed for 10k MNIST training data, with contractive autoencoder projection (see Section 5.2). Color corresponds to projection value, and node size corresponds to total number of points in the node.  $n_{\text{int}} = n_{\text{bins}} = 10$  for the LHS figure,  $n_{\text{int}} = n_{\text{bins}} = 20$  for the RHS figure.

In general,  $\tilde{N}$  will be a function of both  $n_{\text{bins}}$ ,  $n_{\text{int}}$ , the function  $f$ , and the data. The Mapper procedure gives clustering information based on the original data  $X_{\text{train}}$  and the information contained in  $f|_{X_{\text{train}}}$ . In step (3), a local neighborhood scale must be defined or recovered in order to produce a refinement of the open cover  $V_\alpha$ . This is done by first producing a histogram of the number of components that become connected at varying length scales via single-linkage clustering (see Figure 5.3). If there are distinct clusters in the data, the histogram will have at least two main peaks: one peak corresponding to points that become connected at smaller distance scales, and another corresponding to points that become connected at larger lengths which represent the distance between clusters. The heuristic we use to choose the small distance

scale is the value at which the first break in the histogram occurs, which is motivated from the distinction between separate peaks in the histogram that come from different scale properties of the data. This process is repeated for each set  $U_\alpha$  in the open cover, and a new local neighborhood scale is recovered for each of these sets.

A higher bin value will tend to push down the distance scale that is required in step (3), and hence produce more nodes in the Mapper. Thus, a higher bin value can also be seen to correspond to, on average, an increase in the complexity of the Mapper object. In general, there is a bias-variance tradeoff in setting this value. See Figure 5.2 to see how this choice can change the Mappers. Additionally, we fix the gain for the sets in the open cover  $U_\alpha$ , to be a 33% overlap. This will cause each data point to be assigned to at most 2 nodes in the Mapper.



**Figure 5.3:** Histograms of length scales at which clusters are grouped via single-linkage clustering. These plots represent varied  $n_{\text{bins}}$  for the first interval in the open cover of  $U_\alpha$ , using the filter  $f = PCA_1$ . In this case, increasing  $n_{\text{bins}}$  from 10 to 20 has no effect on the cutoff value, which is set to 6.6.

**Remark 5.1** *The undirected graph constructed from the set of vertices  $\{(\alpha, i_\alpha)\}$ , and having edges whenever the intersection of two data components is nonempty, i.e.,  $V_{\alpha, i_\alpha} \cap V_{\alpha', i_{\alpha'}} \neq \emptyset$ , is interpreted as the nerve complex of the dataset  $X$ . In particular, when using a one-dimensional filtration the resulting nerve complex is one-dimensional (composed out of nodes and edges only). This construction is generalized to higher dimensions.*



## 5.2 Training and Testing Procedures

We will refer to our algorithm as a *Mapper Classifier*, or *MC* for short. In our MC algorithm, using the provided dataset  $X_{\text{train}}$ , we construct a *committee of Mappers*, meaning a pair of sequences of Mapper objects and the corresponding filtration functions used to generate them:

$$C(X_{\text{train}}, f) := \left( \{M_j\}_{j=1}^{N_C}, \{f_j\}_{j=1}^{N_C} \right) = (\{M_j\}, \{f_j\}), \quad (5.1)$$

where  $f_j: \mathcal{X} \rightarrow \mathbb{R}$  are the filter functions and  $N_C$  is the number of members chosen to be included in the committee. We emphasize that building a committee of Mappers requires in the first place, some *systematic way of generating filter functions*. For our analysis, we choose the overall filter  $f$  (projection onto a latent space) to be either:

1. PCA [29], where each  $f_j$  is a mapping onto the  $j$ -th principal component which is constructed using  $X_{\text{train}}$ .
2. An autoencoder, which we use several variants of, including: contractive [102], deep [103], and variational [104]. Each  $f_j$  is a mapping onto the latent space generated by  $j$ -th hidden node of the autoencoder. The autoencoder is first trained using  $X_{\text{train}}$ .

We define a map that takes data points in  $X_{\text{train}}$  to the vector representation of the Mapper nodes in a single Mapper  $M$  by:

$$g_M: X_{\text{train}} \rightarrow \mathbb{R}^{N_M}, \quad (5.2)$$

where  $N_M$  is the number of nodes of Mapper  $M$ . The map  $g_M$  has a natural definition for all the data in  $X_{\text{train}}$ , as it sends points to a vectorized binary representation:  $g_M(X_{\text{train}}) \subset \{0, 1\}^{N_M}$ . Each training point  $x \in X_{\text{train}}$  gets assigned by  $g_M$  the binary vector representing the mapper vertices  $\widetilde{V}_{\alpha, l_\alpha}$ , such that  $x \in \widetilde{V}_{\alpha, l_\alpha}$ .

**Example 5.2** Assume that a Mapper  $M$  is composed of 3 nodes. Let  $x_1, x_2, x_3 \in X_{\text{train}}$ , such that

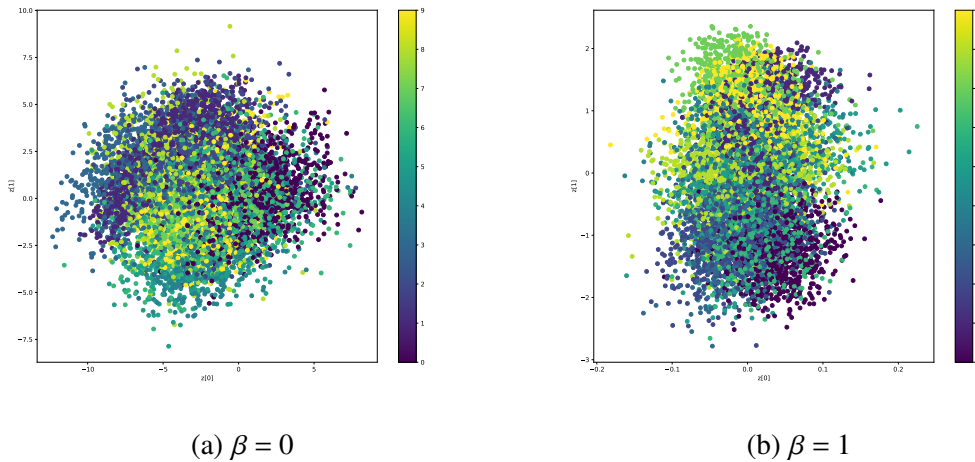
they are in the 1st, 2nd and 3rd vertices of  $M$  respectively. Then,  $g_M(x_1) = (1, 0, 0)$ ,  $g_M(x_2) = (0, 1, 0)$ ,  $g_M(x_3) = (0, 0, 1)$ .

The map  $g_M$  has a natural extension to the Mappers committee  $C$ :

$$g_C : X_{\text{train}} \rightarrow \mathbb{R}^{\sum_{j=1}^{N_C} N_{M_j}}, \tag{5.3}$$

where  $M_j$  represents the  $j$ -th Mapper object in the committee, which consists of  $N_C$  total Mappers, and  $N_{M_j}$  corresponds to the number of nodes in  $M_j$ . This procedure can be seen as joining the vector representations for all the individual Mappers in the committee into a long vector.

The procedure that has been presented for mapping data points in  $X_{\text{train}}$  to the vectorized binary representation applies to the training data only, and for datapoints in  $X_{\text{test}}$  we need to utilize an alternate procedure, that we present in Section 5.2.2.



**Figure 5.4:** Latent space representations of two nodes in the compressed layer of the VAE (i.e., a 2-dimensional subspace of the projection to 20-dimensions). We use a  $\beta$ -term that multiplies the KL divergence.  $\beta = 0$  yields the best overall robustness.

## 5.2.1 Training Procedure

### INPUT:

- The internal parameters of the Mapper algorithm (see Section 5.1.2).
- The number of components ( $N_C > 0$ ) used to build the committee of Mappers. We set  $N_C = 20$  (more on this in Section 5.3).
- The choice in the latent space projection method (either PCA or an autoencoder).
- A split for the training dataset:  $X_{\text{train}} = X_{\text{train}}^1 \cup X_{\text{train}}^2 \cup \dots \cup X_{\text{train}}^n$ , such that  $X_{\text{train}}^i \cap X_{\text{train}}^j = \emptyset$ .
- Labels  $Y_{\text{train}}$  for the training set.

### OUTPUT:

- Classifier for the training set  $X_{\text{train}}$  with labels  $Y_{\text{train}}$ .

### begin

1. Build the committee of Mappers for each data-split  $X_{\text{train}}^i$ :  $C_i = C(X_{\text{train}}^i, f^i) = (\{M_j^i\}, \{f_j^i\})$ .  
Note that each projection  $f^i$  is trained independently on each of the  $X_{\text{train}}^i$ .
2. Using the map  $g_{C_i}$  (see Equation (5.3)) applied to each of the  $C_i$  committees, compute the binary matrix representations of the subsets denoted by  $i$ :  $g_{C_i}(X_{\text{train}}^i) \in \{0, 1\}^{N_i \times \sum_k N_{M_k}}$ , where  $N_i$  is the number of examples in  $X_{\text{train}}^i$ .
3. (only if  $X_{\text{train}}$  is split) using the computed committee of Mappers, compute the ‘off-diagonal’ binary matrix representations, i.e., for all  $i = 1, \dots, n$  compute  $g'_{C_i}(X_{\text{train}}^j) \in \mathbb{R}^{N_i \times \sum_k N_{M_k}}$  for all  $j \neq i$ , using the mapping procedure presented in Section 5.2.2.
4. Train an end classifier (in our case we use a neural network), using the merged data from the previous step. The merged data is represented as a matrix which can be seen as a map  $g(X_{\text{train}}) \in \{0, 1\}^{N_{\text{tot}} \times N_{M_{\text{tot}}}}$ , where  $N_{\text{tot}}$  is the total number of instances in  $X_{\text{train}}$ , and  $N_{M_{\text{tot}}}$  is the total number of nodes over the entire collection of committees, i.e.,  $N_{M_{\text{tot}}} = \sum_i \sum_j N_{M_j}$ .

where  $i$  runs over all data splits and  $j$  denotes specific Mappers in a split. In block form:

$$\begin{bmatrix} g_{C_1}(X_{\text{train}}^1) & g'_{C_2}(X_{\text{train}}^1) & \cdots & g'_{C_n}(X_{\text{train}}^1) \\ g'_{C_1}(X_{\text{train}}^2) & g_{C_2}(X_{\text{train}}^2) & \cdots & g'_{C_n}(X_{\text{train}}^2) \\ \vdots & \vdots & \vdots & \vdots \\ g'_{C_1}(X_{\text{train}}^n) & g'_{C_2}(X_{\text{train}}^n) & \cdots & g_{C_n}(X_{\text{train}}^n) \end{bmatrix}$$

**end**

## 5.2.2 Mapping Unseen Points to the Committee

We describe how we construct the  $g'_{C_i}$  map, a generalization of the  $g_{C_i}$  map (Equation (5.3)) to test datasets (analogously define  $g'_M$  map).  $g'_{C_i}$  is used in order to map test data points to an existing committee of Mappers that is constructed using the  $i$ -th split of data.

The data-points that are being tested through the committee of Mappers can be any set in principle. In the algorithm below, we will denote this set as  $X_{\text{new}}$ , and assume that it is provided as input to our testing algorithm. Examples are  $X_{\text{new}} = X_{\text{train}}^i$  and  $X_{\text{new}} = X_{\text{test}}$  (or some perturbations of data in  $X_{\text{test}}$  as used for robustness testing). We describe in more detail the splitting procedure and its utility in Section 5.2.3.

**IN:**

- The same input information from the training procedure in Section 5.2.1,
- $k \geq 1$ , the number of nearest neighbors considered in the algorithm. We set  $k = 6$  (more on this in Section 5.3),
- the testing data  $X_{\text{new}}$ .

**OUT:**

- Vector committee representation of the new points  $g'_{C_i}(X_{\text{new}})$ .

**begin**

1. Find  $\alpha$ 's such that  $\left| f_j(x) - \text{mid}(U_\alpha^{ij}) \right| < \frac{\max(U_\alpha^{ij}) - \min(U_\alpha^{ij})}{2} (1 + \delta)$ , for  $x \in X_{\text{new}}$ . The  $\delta$  parameter consequently enhances the robustness as it broadens the search space (defined in step 3). The  $\alpha$  denotes the interval in the cover for  $\mathbb{R}$  (see Section 5.1.2),  $i$  denotes the split, and  $j$  denotes the filter (i.e  $j = 1$  for PCA would mean the first principal component).
2. For these  $\alpha$ , collect the corresponding refined vertices  $\widetilde{V}_{\alpha, l_\alpha}^{ij}$ . These are the clusters in the  $i$ -th data split  $X_{\text{train}}^i$  that are mapped to  $U_\alpha^{ij}$  by  $f_j$  and are indexed by  $l_\alpha$ .
3. For all  $x \in X_{\text{new}}$  and for all splits indexed by  $i$  perform the  $k$ -nearest neighbors search within the Mapper vertices found in the previous step  $\{\widetilde{V}_{\alpha, l_\alpha}^{ij}\}$  to find  $nn_1^i(x), \dots, nn_k^i(x) \in X_{\text{train}}$ , where the distance function is chosen to be consistent with the choice of metric used to compute the Mapper committee (Euclidean).
4. For all  $x \in X_{\text{new}}$  and for all splits indexed by  $i$  define:

$$g'_{C_i}(x) = w_1^i \cdot g_{C_i}(nn_1^i(x)) + w_2^i \cdot g_{C_i}(nn_2^i(x)) + \dots + w_k^i \cdot g_{C_i}(nn_k^i(x)),$$

where the weights are defined by  $w_j^i = \frac{[d(x, nn_j^i(x)) + \eta]^{-1}}{\sum_{j=1}^k [d(x, nn_j^i(x)) + \eta]^{-1}}$  where  $\eta = 10^{-5}$ .

**end**

$\eta$  is a stability parameter and does not have a large impact on the results so we opt for setting it to this small value. We use this procedure for any new data point not yet assigned to the  $i$ -th committee, including both  $X_{\text{train}} \setminus X_i$  and  $X_{\text{test}}$ . This process is precisely the  $g'$  map mentioned in Section 5.2.1, and we use it to fill in the missing values of  $g(X_{\text{train}})$  and to construct in its entirety  $g(X_{\text{test}})$ .

The computational complexity of the training and testing portions of solely the Mapper procedure are, in the worst-case scenario,  $\mathcal{O}(n^2)$ , where  $n$  is the number of data points. There are additional computational costs incurred before (i.e., when constructing  $f$ ) and after this procedure

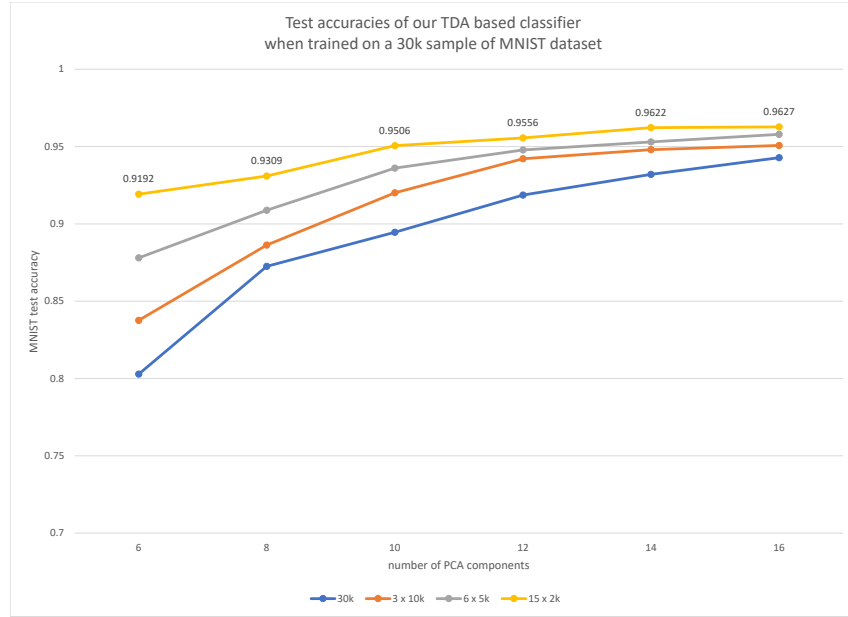
(i.e., when training the end classifier - see Figure 5.1). A lower bound on the complexity is  $\mathcal{O}((n/c)^2)$ , where the  $c$  factor is determined by how points are distributed in the open cover of  $\mathbb{R}$ .

### 5.2.3 Splitting the Dataset

As presented in Section 5.2.1 for the purpose of training and testing we operate on the split training dataset

$$X_{train} = X_{train}^1 \cup X_{train}^2 \cup \dots \cup X_{train}^n.$$

There are two main advantages of splitting the training dataset into subsets. First, it provides a natural way of parallelizing computations during the training/testing phases. This distributed computation procedure is amenable for modern architectures. Computing several small Mapper objects instead of a single large one allows for an easy model by distributing computation among the processor cores. Second, it improves the overall accuracy of the classifier, as illustrated by our results using the MNIST dataset presented in Figure 5.5. The comparison is done using different splittings of a 30k subset of the MNIST training set. Due to computational and memory complexity of the Mapper algorithm, a 30k training dataset is the limit of what we were able to compute using a PC machine having 32gb memory. The run time for 30k was several hours and the memory was fully utilized.



**Figure 5.5:** Test accuracy of MC trained using 30k sample from MNIST dataset. We present the accuracy with respect to the number of PCA filters used in the committee of mappers, and the number of subsets the whole 30k sample was split into.

## 5.3 Numerical Experiments and Results

### 5.3.1 Main Results

In order to quantify overall robustness, we calculate the “normalized accuracy” in Equation (5.4). The main numerical results we report are this accuracy with respect to different noise models for various classifiers, and all are applied to the usual 10k test MNIST data. The noise models we use include: Gaussian blur, random noise selected from a Gaussian, and salt & pepper noise. For additional information on these noise models and how we generated noise, see Section 5.3.3. Additionally, we train on both a 10k subset (examples chosen by random) in the data and the full 60k set. We do not normalize data using std. dev. and mean. For testing, we use the usual 10k MNIST test set. We compute the normalized accuracy as:

$$a(x) = 1 - \frac{\text{missclassified}((0, x])}{\text{initial correct}}, \quad (5.4)$$

where  $\text{missclassified}((0, x])$  is the number of misclassified perturbations within  $l^2$  perturbation norm range  $(0, x]$ . This equation has the benefit that it removes, to an extent, potential dependencies of robustness on the data itself (i.e., we would like robustness to be more a property of the classifier rather than how the classifier interacts with the specific dataset).

There are a few hyperparameters we use, which we will briefly mention here. We set  $N_C$ , the number of Mappers in a committee to 20, because for various classifiers, the initial classification accuracy levels off around this number. By initial classification accuracy, we mean the number of initially correctly classified instances with no noise added. Additionally, we set the number of bins and intervals  $n_{\text{int}} = n_{\text{bin}} = 10$  heuristically, through a combination of what provides a high initial classification accuracy while still uncovering interesting topology as seen by the Mapper objects (see Figure 5.2 and Table 5.2). The  $\delta$  parameter is set to 0.2 (see Section 5.2.2) as this appears to provide the best robustness. Lastly, we set  $k = 6$  when mapping new points to the committee as this provided the best overall robustness for our Mapper classifier (see Section 5.2.2).

We compare our approach based on the Mapper method to the robustness results of a CNN (the standard architecture LeNet [108] was used). The Mapper based methods only differ in their initial projections. For 10k training of MNIST, we investigate Mapper approaches based on: PCA, contractive autoencoder (“CAE”) 784-sigmoid-20-linear-784. For the CAE we include a contractive loss term with a multiplying factor of  $10^{-4}$ . Deep autoencoder (“DAE”): 784-ReLU-1000-ReLU-500-ReLU-250-linear-20 encoder and symmetric decoder; variational autoencoder (“VAE”): 784-ReLU-512-linear-20 encoder and symmetric decoder with sigmoid output. For the VAE, we insert a  $\beta$  term multiplying the KL divergence in the loss [109], and in our case,  $\beta = 0$  provides the most robust results. All these methods can be thought of as projection models to 20-dimensional space (i.e.,  $N_C = 20$ ), and for an example of the VAE projection in



a 2-dimensional subspace, see Figure 5.4. For 60k training of MNIST, we investigate Mapper approaches based on just PCA and VAE since they appear to be the best performing on average. We present the structure of the end classifier that we use in our MC method in Section 5.3.4.

For the initial accuracies of our methods and the traditional CNN approach, see Table 5.1; for the overall robustness calculations, see Figures 5.6 and 5.7.

**Table 5.1:** The initial classification accuracies. PCA, CAE, DAE, VAE are all Mapper based approaches, and only differ in the type of projection. These values give the normalization in Equation (5.4).

	10k Init. Accuracy (%)	60k Init. Accuracy (%)
CNN	97.00	98.51
PCA	94.53	97.33
CAE	93.61	NA
DAE	93.99	NA
VAE	94.95	97.68

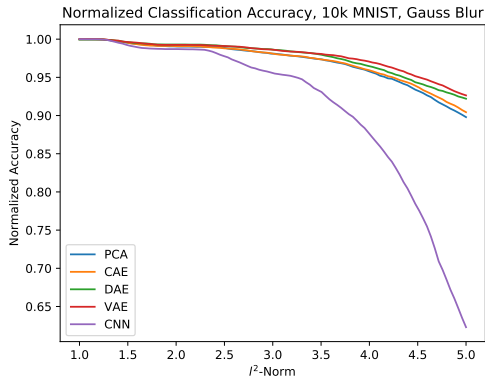
### 5.3.2 Mapper Committee Dimensions

In this section, we summarize the overall mapper dimensions (i.e., total number of nodes in a committee) that data points are sent to. In our analysis, the mapper committee dimension depends on the latent space we project to. In general, this number is highly dependent on  $n_{bins}$  and  $n_{int}$ , but we keep these fixed to  $n_{bins} = n_{int} = 10$ .

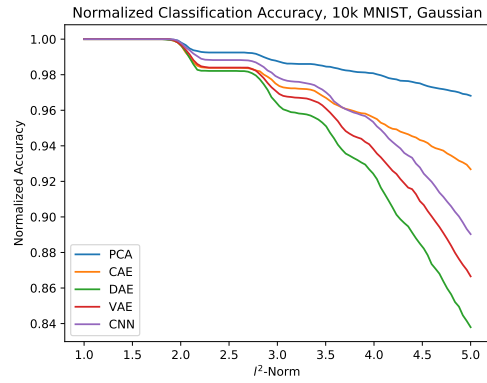
### 5.3.3 Description of Noise Models

#### Models

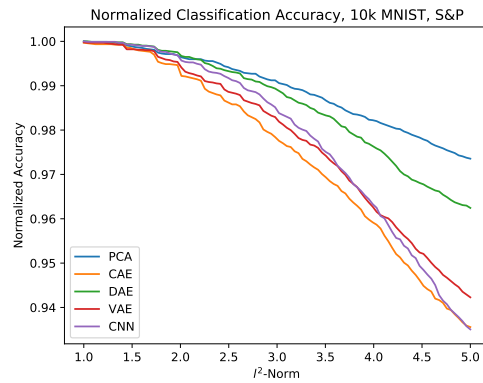
We implement three different noise models in order to determine overall classifier robustness. The models we use include: Gaussian blur, Gaussian, and salt & pepper. All are consistent



(a) 10k MNIST, Gaussian blur

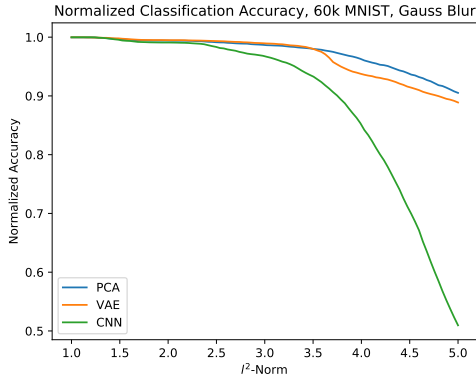


(b) 10k MNIST, Gaussian

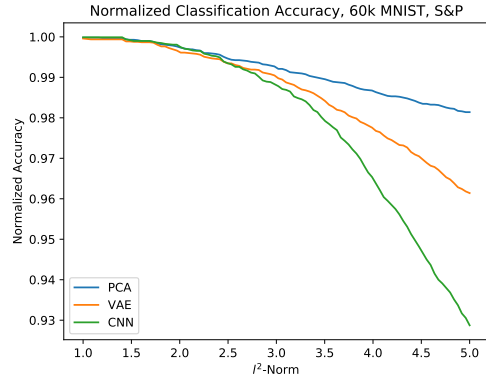


(c) 10k MNIST, Salt & Pepper

**Figure 5.6:** The normalized accuracy with respect to  $l^2$ -norm. The PCA based Mapper approach far outperforms the CNN approach for all the noise models.



(a) 60k MNIST, Gaussian blur



(b) 60k MNIST, Salt & Pepper

**Figure 5.7:** The normalized accuracy with respect to  $l^2$ -norm. The PCA based Mapper approach far outperforms the CNN approach for all the noise models. For 60k MNIST, we choose to investigate just two Mapper based methods: PCA and VAE which seem to perform the best on average.

**Table 5.2:** The total number of nodes in the mapper committee, with respect to choice in latent space projections. For 60k, we choose to use just PCA and VAE since they are the highest performing. Additionally, for 60k MNIST, we use the splitting procedure presented in Section 5.2.2 which will increase the dimensionality of the Mapper committee.

Latent Space	10k	60k
PCA	215	1326
CAE	201	NA
DAE	201	NA
VAE	207	1337

with the models in [110], and we give a brief explanation below. For each of these models, we use a parameter to control the extent of the perturbation, which we refer to as  $\lambda$ .

The *Gaussian blur* model performs a convolution with a 2-dimensional Gaussian centered at each pixel in the image. Each pixel is replaced by the Gaussian weighted sum of nearby pixel values. The  $\lambda$  parameter we use for robustness calculations is related to the Gaussian standard deviation by:  $\sigma = 28\lambda$ . The final perturbed image is clipped to the max and min values of the original image. The 28 comes from the image size of  $28 \times 28$ .

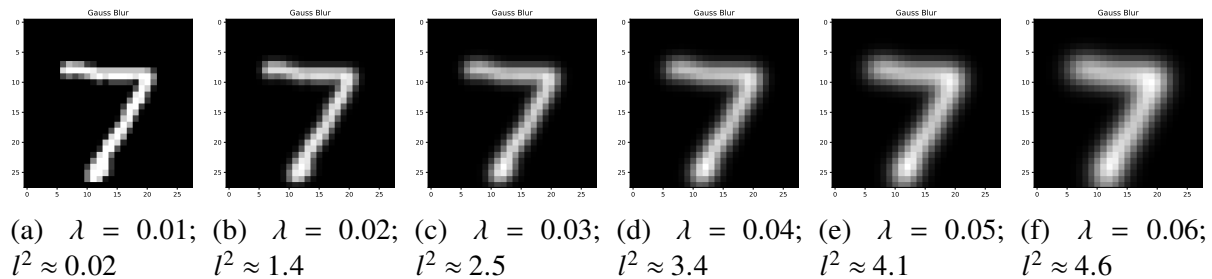
The *salt & pepper* or *s&p* model replaces random pixel values with the minimum (i.e.,

“pepper”) or maximum (i.e., “salt”) in the image. Setting  $q_1$  as the probability of flipping a pixel, and  $q_2$  as the ratio of salt to pepper, we use:  $q_1 = \lambda$  and  $q_2 = \frac{1}{2}$ .

The *Gaussian* model adds in noise to each pixel which is sampled from a Gaussian distribution. The distribution we use is centered at zero and has  $\sigma = 0.1\sqrt{\epsilon}$ . The final perturbed image is clipped.

There are a few subtle differences between the robustness results when  $\lambda$  vs the  $l^2$ -norm is used, which we remark on here.  $\lambda$  is the internal parameter we use to quantify the scale of noise that is added. While the value of  $\lambda$  correlates to the actual  $l^2$  distance an image is perturbed, there is not a one-to-one correspondence. Perhaps a more useful way to think about  $\lambda$  is that it sets a range over which  $l^2$  perturbations may occur. As  $\lambda$  increases, so does this range. Since the  $l^2$ -distance is a more physical measure in this experiment, we report robustness as a function of  $l^2$  rather than  $\lambda$ .

### Image Data as a Function of Noise Parameter

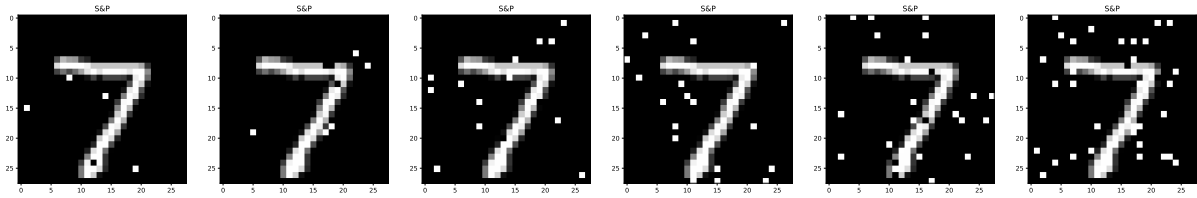


**Figure 5.8:** The number 7 as a function of  $\lambda$  for the Gauss blur noise model.  $\lambda$  can be thought of as a percentage of 28, a fundamental length scale in this data.

### 5.3.4 Architecture of the End Classifier

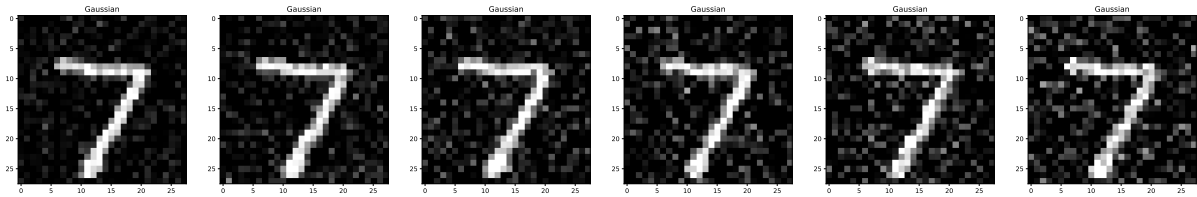
We used the following neural network architecture as the classifier on top of the Mapper method (see Figure 5.1):

1.  $\text{ReLU}(\text{committee } C \text{ dim } \sum_{j=1}^{N_C} N_{M_j}, 4000)$ ,



(a)  $\lambda = 0.01$ ;  $l^2 \approx 2.4$     (b)  $\lambda = 0.02$ ;  $l^2 \approx 2.7$     (c)  $\lambda = 0.03$ ;  $l^2 \approx 3.6$     (d)  $\lambda = 0.04$ ;  $l^2 \approx 4.3$     (e)  $\lambda = 0.05$ ;  $l^2 \approx 4.8$     (f)  $\lambda = 0.06$ ;  $l^2 \approx 5.6$

**Figure 5.9:** The number 7 as a function of  $\lambda$  for the s&p noise model.  $\lambda = 0.01$ , for instance, corresponds to a 1% chance of flipping a pixel.



(a)  $\lambda = 0.01$ ;  $l^2 \approx 2.2$     (b)  $\lambda = 0.02$ ;  $l^2 \approx 2.9$     (c)  $\lambda = 0.03$ ;  $l^2 \approx 3.4$     (d)  $\lambda = 0.04$ ;  $l^2 \approx 3.8$     (e)  $\lambda = 0.05$ ;  $l^2 \approx 4.4$     (f)  $\lambda = 0.06$ ;  $l^2 \approx 4.6$

**Figure 5.10:** The number 7 as a function of  $\lambda$  for the Gaussian noise model.

2. Dropout with  $p = 0.25$ ,
3. ReLU(4000, 2000),
4. Dropout with  $p = 0.25$ ,
5. ReLU(2000, 10),
6. LogSoftmax normalization,
  - The negative log likelihood loss,
  - batch size 100,
  - SGD optimizer with learning rate=0.01, momentum=0.9.

### 5.3.5 Discussion

We present some conclusions from the performed numerical experiments. Our MC method is in general more robust than the CNN, however MC achieves slightly less initial accuracy. We believe that a more extensive hyperparameter optimization of our algorithm would result in a higher initial accuracy. Rather surprisingly, the nonlinear latent space generation using autoencoders performs, on average, worse than the linear PCA method. This is interesting especially because one of the applications of CAE and VAE methods is for adversarial defense by projecting perturbed data onto a small neighborhood in the latent space. Using an autoencoder resulted in better robustness when compared to PCA only in the case of Gaussian blur for 10k training, but this difference is not large. When we extend the analysis to 60k, we see PCA being the clear winner. While we currently do not have a precise answer as to why PCA does so well overall, we expect that the nonlinear methods may be overfitting to the mathematical structure and hence the overall robustness is negatively affected when using them.

Although our MC method performs particularly well for all noise models, it far outperforms the CNN for the case of Gaussian blur noise. This noise model is a global perturbation rather than a local one because it takes a weighted sum of other pixels. Our intuition is that a global perturbation is more likely to change the activation patterns of the feature module in the CNN and hence fool the classifier. We believe our MC methods will do far better than CNNs in general for other global based noises. Geometrically, we take this to mean that there are certain perturbation directions in  $\mathcal{X}$  which are more prone to fooling the CNN; whereas in our MC based approach, the directionality of the perturbation should not matter. A different way of saying this is that the MC approach is noise agnostic, while the CNN is not.

Also, we stop perturbing points when their  $l^2$ -norm becomes around 5 as this range is a more typical “adversarial” range. However, since the MC methods have a smaller slope at the tail ends of the curves, we expect our method to perform even better than the CNN if we were to probe larger perturbation regions, especially since these will approach more global scales.

We will also briefly comment on the flat regions appearing for Gaussian noise Figure 5.6(c). This turns out to be an artifact of our sampling procedure, where we sample noise perturbations scaled uniformly by a “lambda” parameter (see the description in Section 5.3.3). This procedure is not equivalent to sampling among  $l^2$ -norm perturbations (again, see Section 5.3.3).

## 5.4 Theoretical Results

We present a Proposition that formalizes the intuition that our method should be robust with respect to small perturbations of input images.

Intuitively, the presented proposition states that for any training point  $x \in X_{\text{train}}$  it holds that a slight perturbation of the point within range  $\varepsilon$  will also satisfy  $\|g_M(x) - g'_M(x')\|_{l^1} \sim \varepsilon$  (dependence is linear with a small constant). This in turn implies that small changes in inputs will transfer onto slight changes in the  $g_M$  map output. Eventually, the  $g_M$  map outputs are fed into a neural-network based classifier, hence, the eventual robustness is dependent on the precise properties of the employed classifier and how it interacts with the  $g$  map. We denote the  $k$  nearest neighbors of  $x'$  by  $nn_1(x'), nn_2(x'), \dots, nn_k(x')$ .

**Proposition 5.3** *Let  $\{U_\alpha\}$  be the open interval cover of  $I$ . Let  $f: \mathcal{X} \rightarrow \mathbb{R}$  be a Mapper filter function; let  $0 < \delta < 1$  be a parameter of the method (in the actual algorithm  $\delta = 0.2$ ), and let  $k$  be the number of nearest neighbors considered in the algorithm.  $d(\cdot, \cdot)$  is the  $l^2$  distance. Let  $h_k(x') = \sum_{l=1}^k d(nn_l(x'), x')^{-1}$ . To simplify the notation, we denote below  $X = X_{\text{train}}$ .*

*Let  $x \in X$  be perturbed to  $x' \in \mathcal{X}$  such that  $\|x - x'\|_2 \leq \varepsilon$  for some small  $\varepsilon \ll (\mathbb{E}_{x \in X} h_{k-1}(x))^{-1}$ .*

*Let  $g'_M(x')$  be computed using the algorithm described in Section 5.2.2. If for all intervals  $U_\alpha \ni f(x)$ ,  $|f(x') - \text{mid}(U_\alpha)| < \frac{\max(U_\alpha) - \min(U_\alpha)}{2} (1 + \delta)$ , then*

$$\mathbb{E}_{x \in X} |g'_M(x') - g_M(x)|_{l^1} \leq \varepsilon \cdot C(k, \mathbb{E}_{x \in X} h_{k-1}(x)) + O(\varepsilon^2),$$

in particular  $\mathbb{E}_{x \in X} |g'_M(x') - g_M(x)|_{l_1} \rightarrow 0$  as  $\varepsilon \rightarrow 0$ .

### 5.4.1 Proof of the MC Robustness

**Proof 5.4** First, the condition  $|f(x') - \text{mid}(U_\alpha)| < \frac{\max(U_\alpha) - \min(U_\alpha)}{2}(1 + \delta)$  guarantees that the perturbed point  $x'$  is mapped by the filter  $f$  into the same intervals in the open cover as the original point  $x$  (see the corresponding description in the algorithm).

Observe that the assumption guarantees that the nearest neighbor is  $x$ . That is,  $nn_1(x') = x$ , and hence  $w_1 = \frac{1}{\varepsilon \sum_{l=1}^k d(nn_l(x'), x')^{-1}}$ . Therefore, we obtain

$$g'_M(x') = \frac{1}{\varepsilon \sum_{l=1}^k d_l^{-1}} g_M(x) + w_2 g_M(nn_2(x')) + \dots + w_k g_M(nn_k(x')),$$

and for  $j = 2, \dots, k$

$$w_j = \frac{1}{d_j \sum_{l=1}^k d_l^{-1}},$$

where we denote  $d_j = d(nn_j(x'), x')$ . It holds that  $g_M(x) = (1, 0, \dots, 0)$  or  $g_M(x) = (1, 1, 0, \dots, 0)$ , as  $nn_1(x) = x$ . Also we have

$$h_k(x') = \varepsilon^{-1} + \sum_{l=2}^k d(nn_l(x'), x')^{-1} = \varepsilon^{-1} + h_{k-1}(x) + O(\varepsilon).$$

Also,

$$\begin{aligned} w_l &= (d(nn_l(x'), x') h_k(x'))^{-1} \\ &= 1 / (d_l \varepsilon^{-1} + d_l h_{k-1}(x) + d_l O(\varepsilon)) \\ &\approx \varepsilon / d_l \\ &= \varepsilon / (d(nn_{l-1}(x), x) + O(\varepsilon)) \\ &= \varepsilon / d(nn_{l-1}(x), x) + O(\varepsilon^2) \end{aligned}$$



Thus, by the triangle inequality,

$$\begin{aligned}
\|g'_M(x') - g_M(x)\|_{l^1} &\leq \frac{|1 - \varepsilon h_k(x')| \|g_M(x)\|}{\varepsilon h_k(x')} + \varepsilon \sum_{l=2}^k \frac{\|g_M(nn_{l-1}(x))\|_{l^1}}{d(nn_{l-1}(x), x)} + O(\varepsilon^2) \\
&\leq \varepsilon \left( 2h_{k-1}(x) + \sum_{l=2}^k \frac{\|g_M(nn_{l-1}(x))\|_{l^1}}{d(nn_{l-1}(x), x)} \right) + O(\varepsilon^2) \\
&\leq 4\varepsilon h_{k-1}(x) + O(\varepsilon^2).
\end{aligned}$$

In the last inequality above we used the bound  $\|g_M(nn_{l-1}(x))\|_{l^1} \leq 2$ , as  $g_M(nn_{l-1}(x))$  have either one or two nonzero entries equal to 1 ( $x$  is in one or two nodes of the mapper  $M$ ). Finally, taking the expectation we have that

$$\mathbb{E}_{x \in X} |g'_M(x') - g_M(x)|_{l^1} \leq 4\varepsilon \mathbb{E}_{x \in X} h_{k-1}(x) + O(\varepsilon^2),$$

and obviously as  $\varepsilon \rightarrow 0$ , then  $\mathbb{E}_{x \in X} |g'_M(x') - g_M(x)|_{l^1} \rightarrow 0$ .

We computed in practice an estimate for the constant  $C(k, \mathbb{E}_{x \in X} h_{k-1}(x))$  appearing in Prop. 5.3, taking  $k = 6$  (the value used in practice), the empirical expectation of the pairwise distance between points is equal  $\mathbb{E}_{x \in X} h_{k-1}(x) \approx 0.5$ , we obtain that  $C(k, \mathbb{E}_{x \in X} h_{k-1}(x)) \approx 1.5$ .

## 5.5 Conclusion

We have developed an algorithm which performs classification that is both robust to adversarial examples and also highly accurate for non-adversarial examples, resolving, to an extent, the bias-variance trade-off present in many machine learning methods. Although we apply our MC to the task of improving robustness of image classifiers, we expect the algorithm to lend itself well to many other classification tasks in general.

There are various avenues for future work pertaining to this research. One such avenue

is to perform a more extensive hyperparameter search. Many of these were set heuristically, or scanned over slightly, but with little scientific approach on converging to optimal values. This is partly due to time considerations in our algorithm - in order to accomplish this search, we will need to construct our software to be more scalable. Another avenue will be to understand why PCA does so well versus the nonlinear projections in our MC. We expect that the MC is less prone to overfitting when using PCA, but this should be verified. A related path to explore is to determine how the MC methods interact with specific noise models - though we expect the MC approach to be noise agnostic. It appears that the MC does much better for global types of noise than the CNN, but this notion can be made more precise, perhaps from a geometric standpoint. Also related, we would like to extend the  $l^2$  search to larger values of perturbations, where we expect our MC to highly outperform the CNN.

Lastly, we would like to understand further how the choice in architecture of the end classifier impacts the overall robustness. In this analysis, we chose the specific NN architecture that is presented, but indeed we utilized different architectures and even random forests at points. We would like to understand more broadly how this classifier interacts with the Mapper procedure.

## **Acknowledgment**

Chapter 5, in part is currently being prepared for submission for publication of the material. Jacek Cyranka, Alex Georges, and David A. Meyer. The dissertation author was a primary investigator and author of this material.

# Chapter 6

## Conclusion and Future Directions

In this dissertation, we have seen to an extent how topological methods in data science can be combined with more conventional machine learning methods in order to produce robust algorithms. In Chapter 4, we saw an example of how persistence can be used as an initial preprocessing step before classification in order to correctly identify extremely noisy data. In Chapter 5, we saw an extension of this work with Mapper objects instead of persistence as the initial preprocessing step.

In either case, the utility of using topological methods was clear from a theoretical and practical viewpoint. The theoretical justifications include: compressed representation, coordination invariance, and deformation invariance. From the practical viewpoint, we saw how extremely accurate classifiers can be produced. In the case of persistence, we constructed a classifier that's 75% accurate for noise that's on the order of the image size; for Mappers, we constructed a classifier that retained  $\sim 95\%$  classification accuracy, whereas the state-of-the-art CNN had an accuracy of  $\sim 60\%$  for the adversarial regions we probed.

The Mapper based classifiers in Chapter 5 are an extremely promising avenue for future work given the extent of their robustness. There are a few questions that went unanswered by the end of that analysis which merit further exploration. A few of these include: what is the reason

behind the interactions we saw between the Mapper classifier and specific noise models, how does the Mapper procedure interact with the rest of the workflow, how does the classifier overall perform for larger perturbation regions, and why is PCA the optimal mapping in this approach?

Mapper classifiers are a novel based approach that have been shown to be much more robust than current state-of-the-art methods. It is my hope that they continue to be studied and applied further.

# Appendix A

## Feature-Based Algorithm Selection for Mixed Integer Programming

Mixed integer programming is a versatile and valuable optimization tool. However, solving specific problem instances can be computationally demanding even for cutting-edge solvers. Such long running times are often significantly reduced by an appropriate change of the solver’s parameters. In this chapter we investigate “algorithm selection,” the task of choosing among a set of algorithms the ones that are likely to perform best for a particular instance.

The title of this could more colloquially be referred to as “Teaching a Computer to Problem Solve,” since the overall approach taken here can be applied to more general workflows. The general idea is analogous to how humans are confronted with and solve problems. When we are confronted with a problem, we typically have a number of ways to solve it. What is most import to us in the problem solving (perhaps it is time, money, glamour) will guide how we go about the solution. Computers, on the other hand, typically need to be directed how to approach a problem. This project teaches a computer how to problem solve, in that it will eventually be able to choose how to solve an incoming problem by itself.

In our case, we treat different parameter settings of the MIP solver SCIP as different

algorithms to choose from. Two peculiarities of the MIP solving process have our special attention. We address the well-known problem of performance variability by using multiple random seeds. Besides solving time, primal dual integrals are recorded as a second performance measure in order to distinguish solvers that timed out.

We collected feature and performance data for a large set of publicly available MIP instances. The algorithm selection problem is addressed by several popular, feature-based methods, which have been partly extended for our purpose. Finally, an analysis of the feature space and performance results of the selected algorithms are presented.

## A.1 Introduction

### A.1.1 Mixed Integer Programming

Throughout this paper we present results derived from instances of mixed integer programming (MIP). For our purposes, a mixed integer program can be represented as:

$$\operatorname{argmin}_x \left\{ c^T x \mid b_1 \leq Ax \leq b_2, l \leq x \leq u, x_i \in \mathbb{Z} \forall i \in \mathcal{J} \right\}. \quad (\text{A.1})$$

Here, the variables  $x_i$  are constrained to take integer values for all  $i$  in the variable subset  $\mathcal{J} \neq \emptyset$ . An integer variable is called a *binary variable* if,  $l_i = 0$  and  $u_i = 1$ .

Solving a MIP instance (or even determining if a solution exists) is  $\mathcal{NP}$ -hard in general, and there are a variety of competitive solvers available that use different methods and approaches. These solvers are generally used as stand-alone applications and permit the user to specify a large number of parameters which influence the method/approach that the solver uses. For instance, parameters may affect the time spent in presolving, the aggressiveness with which cutting planes are used or the amount of reliance on branch and bound techniques. In their work, Hutter et al [111] describe one of the leading commercial solvers (CPLEX) as having more than  $10^{45}$  distinct

combinations of parameter settings possible (this number has likely changed since the paper was released).

This leads to a number of questions, perhaps the most important of which is what combination of parameters should be used to solve a MIP instance. Arguably, one should use the parameter settings that have the best mean performance on some benchmark set. However, the results that one can achieve in this manner have a limit to their effectiveness, and to overcome this limit one must pay more attention to the properties (features) of an individual instance and allow these features to guide the parameter selection, which is commonly addressed as *Algorithm Selection*.

### **A.1.2 Algorithm Selection**

The underlying nature of MIPs remains an area of open research, and so is not fully understood in terms of complexity. There is no solver that is optimal for a wide class of problems. While it is possible to choose an algorithm with the best average performance on a representative benchmark set, being able to select from a portfolio of algorithms has been shown to be better in cases where complementary, fundamentally different strategies are available [112]. In algorithm selection, this idea is taken one step further by constructing custom portfolios using instance-specific features [113]. To this end, we use features specific to the problem instance to select an algorithm (or a portfolio of algorithms) that are likely to perform well on the given instance.

Algorithm selection has been gaining attention in the artificial intelligence community. To aid in this task, recent libraries have been established that compile and organize data for a wide range of  $\mathcal{NP}$ -hard tasks [114]. While approaches to the algorithm selection problem have been effective for the satisfiability problem, algorithm selection for mixed integer programming has still produced only modest results [113]. It is on this last point that our research is focused.

In the present work, we focus on the algorithm selection problem using various algorithms contained in the SCIP Optimization Suite [115], which is one of the leading non-commercial

software packages that can solve MIP instances. We use 14 different combinations of parameters supplied by SCIP. These parameter combinations serve as the algorithms that we will be selecting from. While we restrict ourselves to the collection of parameter settings suggested by SCIP, we note that due to the highly parameterizable nature of the solver, the number of considered algorithms can be easily increased.

We test the different algorithms on two different benchmark sets, one homogeneous and one heterogeneous. General features of the benchmark instances are obtained from the MIP representation (A.1) to inform our selections and compare the predictive results of different machine learning engines trained on the benchmark runtime data.

The MIP instances within our benchmark sets are supplied in MPS format. We use the SCIP interface to load them and then extract the features, which will be described in Section A.2 and in full detail in Appendix A.7 of this document.

### **A.1.3 Related Work**

Algorithm configuration of MIP solvers has been widely discussed and researched (see [111, 116] and references therein), the main task being to produce solvers that have good average performance on a particular benchmark set. These solvers should then be good predictors of algorithms to use for instances that are similar to those within the benchmark.

Leyton et al. [112] demonstrate the effectiveness of using a portfolio approach in which several algorithms are used either by a schedule or in parallel rather than restricting the solving process to only a single algorithm. This approach has since become commonplace as seen in [113, 117, 118]. We will also use a portfolio approach in our work, selecting more than one algorithm in order to improve performance when the average-best solver does not perform well on a specific instance.

Algorithm selection has proven effective for SAT instances (in particular, for SAT competitions), but has not yet had the same success for MIP instances [118]. Why this is the case is still



an open question.

In [116, 119], combinatorial auction problems are used as the benchmark set for testing solver efficiency. Some previous literature has explored the feature space [111] as well as an attempt to use features to select algorithms. However, we desire to understand the problem in the context of a wider benchmark set with a wider representation of possible features. We also implement the algorithm described in [117] together with some variants of the approach.

## A.2 Method

### A.2.1 Description of Data

Every MIP is characterized by a vector of numeric properties, so-called features. The complete feature data is represented as an  $n \times m$  matrix  $\mathcal{F}$ , where  $n$  is the number of MIP instances under consideration and  $m$  is the total number of features extracted from each problem instance. The performance data (referred to as  $\mathcal{P}$ ) consists of an  $5n \times s$  matrix, where  $s$  is the total number of settings used. The factor 5 here comes from running each solver on every instance 5 times, each with a different random seed specified on the run. The random seed serves as a final tie-breaker for numerous algorithmic components within SCIP. The intention behind seeded runs is to make the performance evaluation between settings more robust in the light of the phenomenon of performance variability [120, 121]. We take the arithmetic mean of results over these random seeds to smooth out statistical fluctuations produced by each run.

What is more, the data comes from two distinct sets: `Regions` which is a collection of 2000 combinatorial auction instances [119], and a general MIP testset called `M&C` with 713 instances compiled from the publicly available libraries `MIPLIB 3` [122], `MIPLIB 2003` [123], `MIPLIB 2010` [124], and `COR@L` [125]. Each of these sets has their own performance data which was computed on an HPC cluster, and we compute feature data independently for each of these sets.

While each instance from the `Regions` set was feasible and reasonably sized, it was necessary to exclude some instances from the `M&C` set. For our purpose, and in particular for our desired experiments with the primal-dual integral (which does not have substantial qualitative meaning for infeasible instances), we excluded the 93 infeasible instances from the `M&C` library. We also excluded eight instances for which features could not be extracted without exceeding the amount of memory available on a desktop computer. This leaves 612 feasible instances in the `M&C` benchmark set with which we conduct our experiments.

The `M&C` library seems to be more challenging for the machine learning methods as it is a heterogeneous set of problems and in addition, it contains fewer instances than the `Regions` library.

## **Feature Data**

The idea of using features from MIPs for algorithm selection was first explored in [113]. As an example, features can be statistical summaries of the coefficients of the objective function, “right-hand side” vectors and the constraint matrix itself. Features extracted in such a way are generally called ‘static’ features since they can be extracted without invoking the operant functions of the solver. We use some static features introduced in [113] as a basis for our static feature calculation.

We extract 133 static features for each problem instance. The list of all static features that we extract can be found in Appendix A.7 along with a description. Since presolving can significantly improve the performance of the solving process, we also extract static features after presolving the problem instance with SCIP’s default and fast presolve settings, which leads to a total number of 399 static features for each problem instance.

With our benchmark instances, we also extract “dynamic” features which include information extracted from the log files of running SCIP with default settings on a particular instance. We limit such extraction to features that are available up to and including the solving of the

root node relaxation. In the larger context of algorithm selection, it is our expectation that this dynamic feature extraction at the root involves only a minimal time commitment relative to the time involved to solve a difficult instance to completion.

## **Performance Data**

For each instance in our benchmark sets, and for each of the 14 different algorithms, we have the following data listed below. To increase robustness, we have averaged the data over five runs for each instance/algorithm. If any run for a particular instance/algorithm takes longer than 600 seconds, we terminate the run early and report the information accumulated up to that point.

- **Timing Data**

The solving time is the most important performance data for instances on which the solver did not run into the time limit. For the homogeneous `Regions` benchmark set, with few exceptions, SCIP was able to solve the instances within the given time limit. For the more heterogeneous set `M&C`, the timing data is much more varied, with a much larger number of timeouts, and in fact, some instances timed out regardless of the algorithm chosen.

- **PD-Integral**

As a second measure of performance, the integral of the primal-dual gap was used. Herein referred to as the primal-dual integral, this measures jointly the quality of solutions found during the solving process as well as the efficiency with which these solutions are found. Particularly it is the percent gap between the best known upper and lower bounds on the true solution found in the branch and bound tree, integrated over time. This measure was introduced in [126] and does not require any prior knowledge of the ground truth solution for an instance. In our research, we investigated this value and its usefulness as a suitable proxy for runtime when the runtime information was not available (for instance,

if a particular run was cut off early by timing out). For further information on how this integral is explicitly calculated, we refer the reader to [126].

- **Completion Codes**

As a final categorical measure of performance, we have the exit codes provided by SCIP corresponding to various forms of success/failure of the solver on a particular instance/algorithm attempt. We have aggregated these completion codes into the three coarser categories of ‘ok’, ‘timelimit’, and ‘fail’ (details of this aggregation are given in Appendix A.8). ‘Ok’ corresponds to a successful run of the algorithm and no data adjustment is necessary. ‘Timelimit’ corresponds to an algorithm that went beyond its allotted time, in which case we take the execution time to be 600s. ‘Fail’ corresponds to a situation in which either the solver failed to complete (e.g. due to memory limits), or delivered an answer that was not satisfactory (e.g. claiming optimality of a suboptimal solution). If the solver failed, we take the execution time to be 600s and the PD Integral to being 60,000 (corresponding to 600s of 100% gap).

## **A.2.2 Upper Bounds/Well-Defined Features**

Within the benchmark set, there can be instances for which numeric data is not well-defined. For instance, if an instance has no left bounds in its constraints, then no average value of the left bounds can be found. In such cases, we use zero.

Additionally, within the timing data, it is possible for an algorithm to complete an instance early but incorrectly (e.g. exceeding the available memory). In such cases, we set the numeric time to completion as 600 seconds (which is our time limit) and the PDI to 60,000 (which is the maximum possible PDI). We note that this presents a challenge for our regression engines because such values are not chosen quantitatively but qualitatively, and it is likely that this choice favors

certain classes of machine learning techniques that utilize branching and decision boundaries over continuous techniques that function primarily on regression.

## A.3 Feature Analysis

As stated previously, we currently extract 399 static features and 103 dynamic features. The static and dynamic features can be considered independently from one another or simultaneously. Since the dimensionality of the feature space is relatively large in either case, a few natural questions that arise are:

**Q1** Which features are the most important?

**Q2** Which features are the least important?

**Q3** Is there a lower dimensional representation of the feature space that accurately encapsulates most of the information.<sup>1</sup>

We devote the majority of our attention in answering **Q1** and **Q3** by using Pearson correlation coefficients, principal component analysis, and multidimensional scaling.

### A.3.1 Pearson Correlation Coefficients

Pearson correlation coefficients are normalized values of the covariance matrix, and they take on values between +1 and -1. A score of +1 indicates that two variables are positively linearly correlated, and a score of -1 indicates they are negatively linearly correlated. We compute this via:

$$\rho(x, y) = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y}$$

---

<sup>1</sup>This is defined in more detail in a later section

Here,  $cov(x, y)$  denotes the covariance between  $x$  and  $y$ , which are features in  $\mathcal{F}$ . Also,  $\sigma_x$  and  $\sigma_y$  denote the standard deviation in  $x$  and  $y$ , respectively. We plot a heatmap of these correlation coefficients, which allows us to qualitatively answer a few questions:

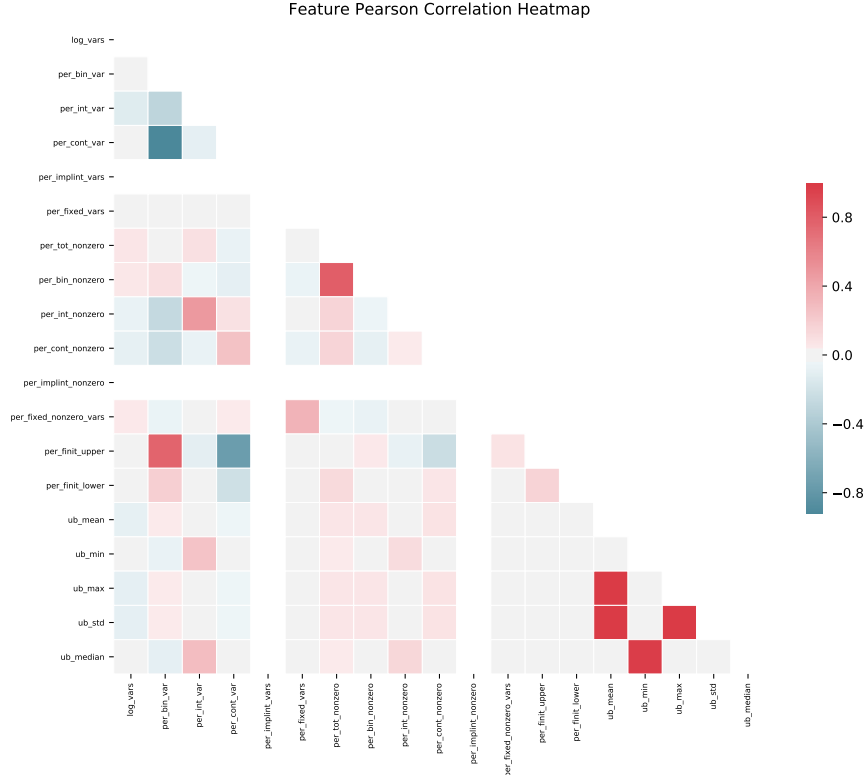
- Are there any features which can be removed from the feature space? Features which have many correlations, whether they are positive or negative, have their information more or less encoded in other features. These highly correlated features can be removed from the space because of this, thus answering **Q2**.
- What features are important and why? This is most useful when used in conjunction with another approach. For instance, we utilize various techniques that predict which features are the most important. The heatmap can be a useful tool in answering why are they the most important, thus answering **Q1**.
- Are there any issues in the features themselves? For instance, does a particular feature fail to vary within our benchmark set?

As an explicit example, we produce the correlation heatmap using the first 20 dimensions in the feature space (Figure A.1). In this case, the feature with zero correlation turns out to be a feature which can be removed from the features space as it renders no useful information. Specifically, this feature was zero across all instances.

### A.3.2 Principal Component Analysis

We use the same PCA approach presented in Section 3.3 to study the feature space, and we similarly plot the information as a function of  $d$  in Figure A.2.

PCA additionally allows us to answer the question of “what features are the most important?” The components of  $\hat{\mathcal{F}}$  are constructed through linear combinations of features in  $\mathcal{F}$ . By determining which features are the highest weighted across all linear combinations, we are able to



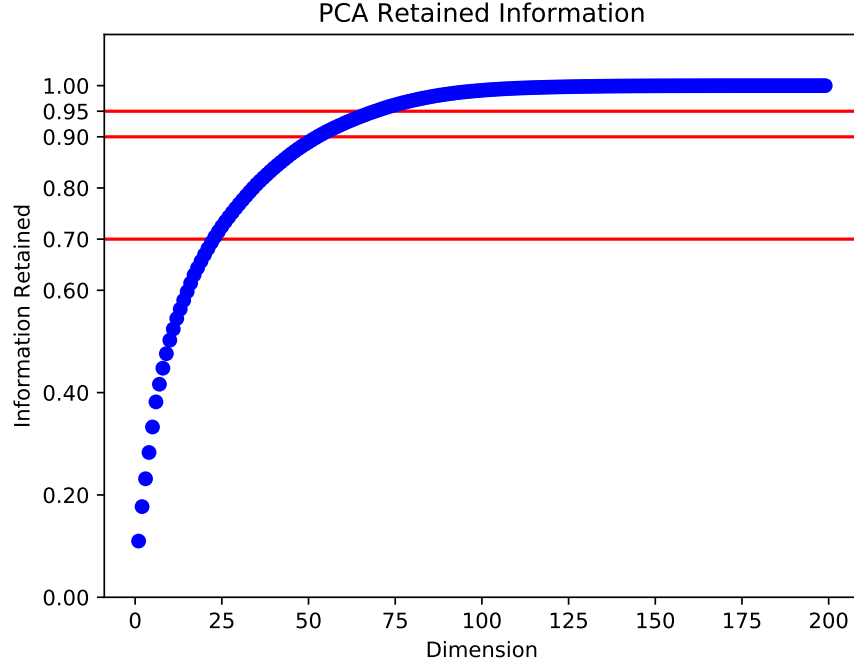
**Figure A.1:** A Pearson correlation coefficient heatmap computed using the first 20 dimensions in the feature space. The diagonal and upper triangular elements have been removed since this heatmap is a symmetric plot.

select a list of individual features which PCA relied most heavily on to construct  $\hat{\mathcal{F}}$ . By keeping 95% of the PCA information<sup>2</sup>, we select 71 features.<sup>3</sup> Thus, we answer both **Q1** and **Q3** using PCA.

We note a peculiar clustering that occurs in  $d = 2$  when we consider just the dynamic features (refer to Figure A.3). When we consider all features, or just the static features, we obtain no similar clustering (Figure A.4).

<sup>2</sup>i.e., setting  $I(d) = 0.95$

<sup>3</sup>We point out that although  $d = 71$  refers to 71 dimensions in PCA space, and *not* in the original feature space, we nonetheless use this as motivation to select 71 features from the original space.



**Figure A.2:** The PCA retained information as a function of dimension given by Equation (3.1). The input data is the static+dynamic feature space. The various horizontal lines are 70%, 90% and 95% thresholds. The 95% threshold occurs at  $d = 71$ .

### A.3.3 Multidimensional Scaling

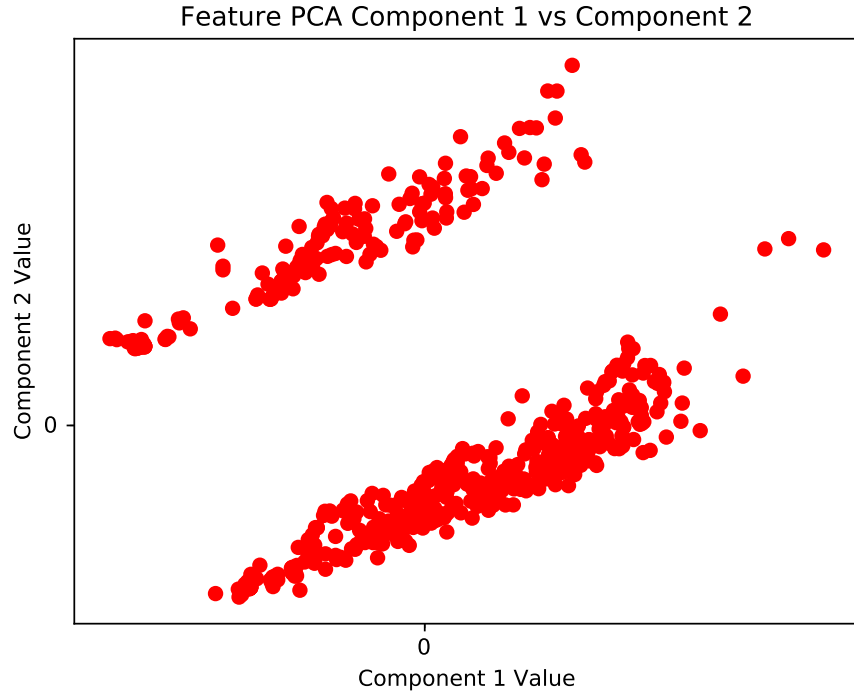
Similar to PCA, multidimensional scaling (“MDS”) is a dimensional reduction technique, but it fundamentally differs in that it does so nonlinearly. Rather than attempting to maximize the variances as PCA does, MDS tries to preserve pairwise distances as well as possible in a lower dimensional space. How well the method does can be quantified through an error<sup>4</sup> function which depends on the dimension being mapped to:

$$err(d) = \sum_{i \neq j} |d_{ij} - \hat{d}_{ij}(d)|^2$$

Here,  $d_{ij}$  is the pairwise distance between the  $i$ -th and  $j$ -th points in the original space,  $\hat{d}_{ij}$  is the pairwise distance in the lower dimensional space and  $d$  is the dimension being mapped to.

<sup>4</sup>The loss function used in MDS is typically referred to as the “stress”





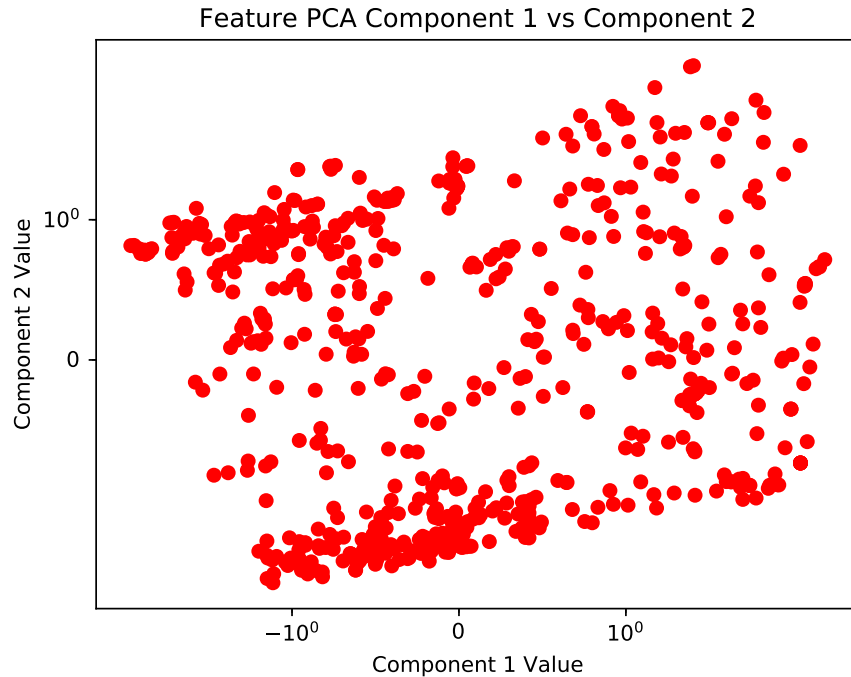
**Figure A.3:** PCA components 1 and 2, for just the dynamic features for M&C. Notice the distinct 2 clusters that emerge.

The goal in MDS is to minimize  $err(d)$  by choosing an appropriate  $\hat{d}_{ij}$ . Similar to PCA, we also define a measure of the retained information when mapping to a lower dimension:

$$I(d) = 1 - err(d)/err(2) \tag{A.2}$$

This form was chosen so that the information is constrained to  $[0,1]$  and so that it monotonically increases with respect to the dimension. Additionally,  $I(2) = 0$  and  $I(m) = 1$ . Setting a 95% threshold on the retained information results in  $d = 19$  (refer to Figure A.5). Using this technique we are also able to answer **Q3**.

Similarly, we note a peculiar clustering that occurs for just the dynamic features when the MDS projection is done for  $d = 2$ . See Figures A.6 and A.7. We aim to identify the source of these clusters in future work. Currently, we posit that each cluster will correspond roughly to two classes of problem difficulties.



**Figure A.4:** PCA components 1 and 2, for static and dynamic features for M&C. Notice, the cluster information is washed away.

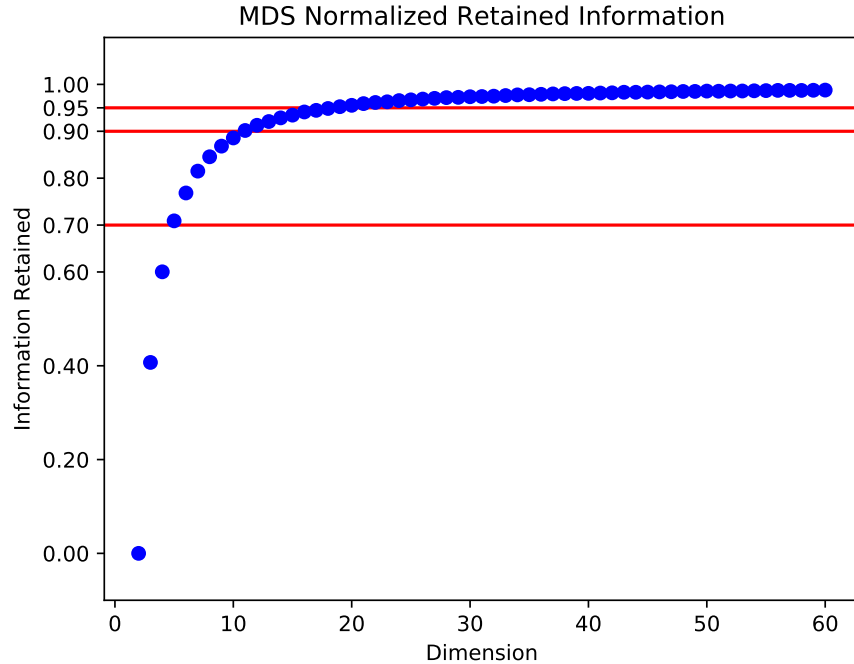
### A.3.4 Feature Investigation Conclusions

We summarize a few key observations and results to answer our original questions about the feature space.

We break up our first question (**Q1**) about the feature space into two questions: **Q1a** which features are the most important to a specific model and **Q1b** which features are the most important in general? The answer to **Q1a** simply depends on the model. For instance, the heuristic for selecting the top features for PCA has already been addressed. The heuristic for selecting the top features for a Random Forest Regressor or Classifier depends on which features result in the best tree splits. More specifically, each feature results in a split with a certain amount of information gain (in the case of a decision tree) or error reduction (in the case of a regression tree). The splits with the most<sup>5</sup> information gain or the least error over the entire forest become the top features.

---

<sup>5</sup>Here, “most” can be set in various ways, but it more or less means higher than some multiple of the average



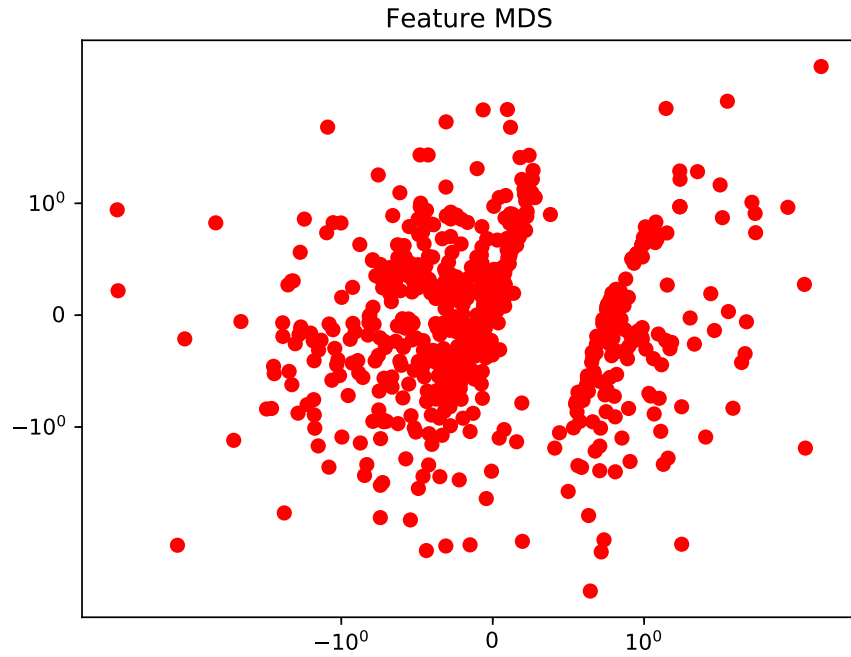
**Figure A.5:** The MDS retained information as a function of dimension given by equation (A.2). The input data is the static+dynamic feature space. The 95% threshold occurs at  $d = 19$ .

To answer question **Q1b**, we take the intersection of all the model dependent results to construct a more model independent result:

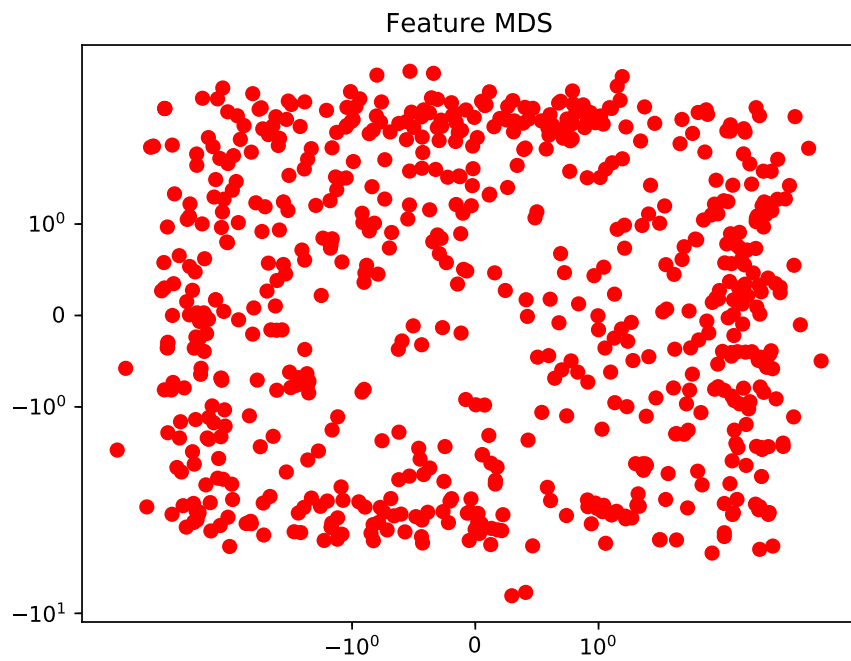
$$\text{Top Features} = \bigcap_i \text{Top Features}_{\text{model}_i}$$

The models we currently use that go into this intersection are: a random forest regressor, random forest classifier that utilizes Hydra (see below), decision tree classifier that utilizes Hydra (see below), and PCA. The full list of model independent top features for the M&C library is available in Appendix A.7.

To answer **Q3** (i.e., the question of the true dimensionality of the feature space), we look at the number of each model dependent top features list and the information content of PCA and MDS (refer to Figures A.2, A.5). The true dimensionality of the feature space appears to be, on average, around 15% of the dimensionality of the total feature space.



**Figure A.6:** MDS projection to  $d = 2$ , for just the dynamic features for M&C. Notice the distinct 2 clusters that emerge.



**Figure A.7:** MDS projection to  $d = 2$ , for static and dynamic features for M&C. Notice, the cluster information is washed away.

## A.4 Algorithm Selection Methods

Three distinct methods have been used extensively in the algorithm selection community to predict which algorithms might be the most successful: those based on classification, regression, and clustering. We consider machine learning methods that are based on all these approaches. However, we note the success of random forest regressors for the task of algorithm selection [118]. More specifically, we apply random forest regressors, random forest classifiers, support vector machine classifiers,  $k$ -nearest neighbours, and logistic regressors. In Section A.5, we present the best performing methods only.

For some of the learning methods, we choose to implement additional boosting methods: Adaptive Boosting (AdaBoost) [127] or Hydra [117]. Each of these methods trains on a set of weak learners, the output of which is a weighted sum of these weak learners. Additionally, we tune various parameters independently for each method in order to minimize the testing error. To reach maximal performance, a future investigation may be to tune these parameters without this independence assumption. Again, we only present methods that were the highest performing.

### A.4.1 AdaBoost

AdaBoost [127] learns a weighted combination of weak learners in multiple iterations. The motivation for this is to build a strong learner from weak learners. In other words, each weak learner is biased, and AdaBoost is a method to produce minimal generalization error (see Section 2.3.2 for a further explanation of the “bias-variance dilemma”). In each iteration, it selects a weak learner that minimizes the weighted error on the training data. The weights of misclassified samples in the training data increase in the next iteration. In the end, weights of the selected learners themselves are adjusted to minimize the total error of the ensemble’s prediction. We use AdaBoost in conjunction with random forest regressors, and we compare these boosted random forests to the non-boosted versions.

## A.4.2 Hydra

Hydra is a learner voting approach which was originally described by Xu et al. [117]. Their approach uses cost-sensitive decision forests as a decision mechanism and a voting schema that summarizes the results. The experiments here were conducted using a reimplementation of the algorithm as described in the original article. Additionally the algorithm was extended to use other decision-making mechanisms within the general voting part. Particularly decision trees,  $k$ -nearest neighbours, support vector classifiers, and logistic regression algorithms were implemented and evaluated.

For the set of algorithms  $\{s_1, \dots, s_m\}$  Hydra constructs a set of  $m(m-1)/2$  pairwise decision-making mechanisms. For a pair of algorithms  $i, j$  a mechanism  $DM(i, j)$  is trained to decide whether algorithm  $i$  or  $j$  is ranked higher on a given instance. To perform the algorithm selection for an instance, each  $DM(i, j)$  decides which algorithm is better and grants one vote to it, so in the end,  $m(m-1)/2$  votes are distributed. Since a larger number of learners is trained for each run of Hydra, the computational effort is higher than for the other approaches presented here. Also Hydra does not predict actual running times but only a ranking of the algorithms.

## A.4.3 Selecting a Portfolio vs selecting an algorithm

Often, there exist subsets of MIP instances for which a single algorithm performs particularly good or bad. It has been shown in [112, 117, 118] that identifying several algorithms (called a ‘portfolio’) which are used to solve the instance simultaneously, increases the likelihood to conclude in a short amount of time. Following this trend, we also predict not only one best algorithm but a portfolio of algorithms that covers the entire test bed better than any single algorithm.

#### A.4.4 Performance Metric

With our performance data, we want to compare a series of values which can be somewhat disparate or irregular in their distribution. Using the arithmetic mean would over-emphasize outliers, and using the geometric mean would over-emphasize the ratios between small numbers. In [128], Achterberg identified the shifted geometric mean as a suitable compromise between the two approaches, and so it provides a more robust measure of performance for algorithm selection. For a vector  $\vec{x} = (x_1, x_2, \dots, x_n)$ , the shifted geometric mean is given by

$$\sigma_s(x) = \left( \sqrt[n]{\prod_{i=1}^n (x_i + s)} \right) - s$$

where the *shift value*  $s$  reduces the influence of observations close to 0.

In order to calculate the performance of the algorithm selection mechanism on a test set, we calculate the shifted geometric mean of the best performing algorithm in the portfolio predicted by the mechanism. The mean is taken over all of the test instances. The values of  $x_i$  are taken from the real performance data (Primal-Dual integral or solving time as it is described further) and then compared with other algorithm selection mechanisms. In our work we take  $s = 10$  for time data and  $s = 1000$  for PD-Integral data.

Additionally, since we are also making predictions on portfolios of algorithms, in the case that more than one algorithm is selected, we use the value that corresponds to the best of the portfolio. The best number for each instance is taken to be the value of the portfolio, and then the shifted geometric mean over the instances is calculated. As a basis for comparison, we use a portfolio which is constructed without using the feature data. Using the timing data available, we place algorithms into the portfolio preferentially based upon their shifted geometric means over the instances. We refer to this as *featureless algorithm selection*.

In order to compare the performance of two algorithms to each other, we calculate the shifted geometric mean over the instances separately for each algorithm, and then compute the

following performance metric<sup>6</sup>:

$$p_s(f, g) = 1 - \frac{\sigma_s(f(X))}{\sigma_s(g(X))} \quad (\text{A.3})$$

This formula gives us the relative improvement of portfolio producer  $f$  versus portfolio producer  $g$ . In our results, we take  $f$  to be predictions made from our various machine learning methods (i.e., predictions based on both performance and feature data) and  $g$  to be the portfolio produced by featureless algorithm selection.

## A.5 Results

### A.5.1 Performance

We present the highest performing prediction engines we found for the `M&C` benchmark. Recall that this benchmark set is heterogeneous, and while this makes prediction more difficult, we feel that this makes the results more readily generalizable, unlike the `Regions` library which is a homogeneous benchmark and should not be expected to represent the vastly larger MIP instance space. The diagrams we present here are generated using the `M&C` set, as the results from the `Regions` set were quite similar and so for conciseness we present the sole benchmark.

Going into each of these methods is a choice in: the input data, the machine learning method, and whether to boost the learning or not. Random forests were the highest performing methods and inherent to these is a random seed that is used for training. Additionally, a random seed is used for the test/training split (20%/80% in our case). Since there are these sources of randomness, we submit 100 independent runs, where each run is trained on a different seed (i.e., we produce 100 different portfolio predictions for each learning method we utilize). For each run, we compute the performance  $p_s(\text{random forest prediction, default})$  of our prediction,

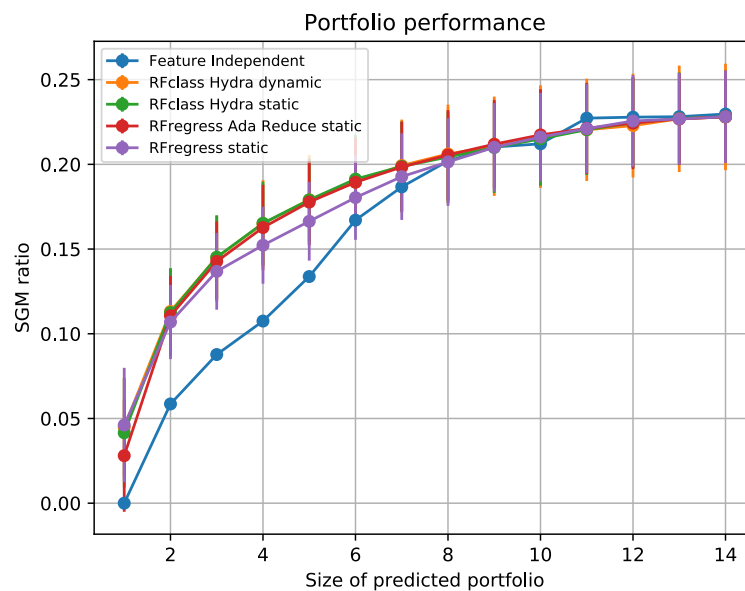
---

<sup>6</sup>The performance metric, which we use to measure the quality of our predictions, is not to be confused with performance data!



cf. Equation (A.3).

We compare the portfolio predictions produced from the random forest to the portfolio produced by always selecting the default solver. We also produce a portfolio from a featureless approach in which we rank each solver by the shifted geometric mean, and iteratively select the best performing solvers (i.e., with the lowest shifted geometric mean). This featureless approach is also compared to always selecting the default solver. In either case, the quantity  $p_s$  can then be thought of as the percent improvement over always choosing the default solver. For example, a value of  $p_s = 0.05$  means that the random forest is choosing a portfolio, the solvers of which will run 5% faster than choosing the default solver.

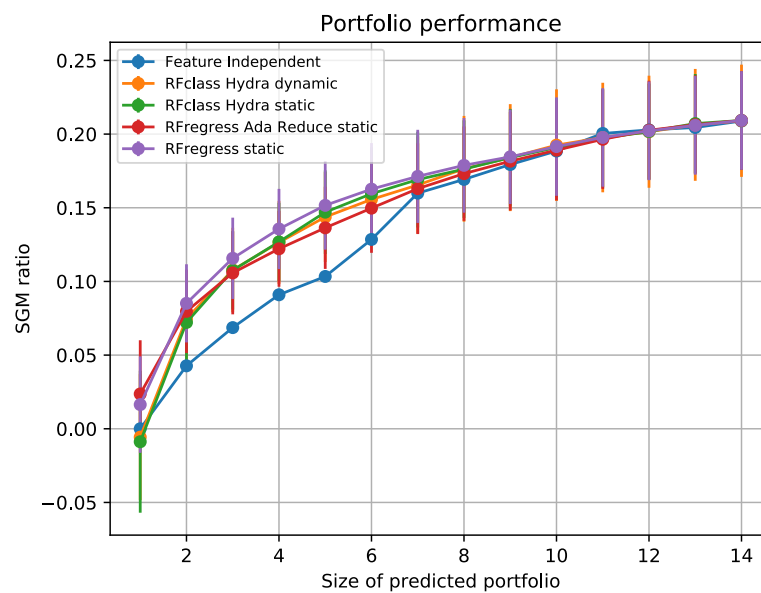


**Figure A.8:** Portfolio performance on the M&C data set. Selectors were trained and tested on Primal-Dual integral values.

We present our results by taking the arithmetic mean of the 100 different  $p_s$  values coming from the independent runs over seeds, and we plot the standard deviation to show the variance of the performance. Using our machine learning methods, we are able to select algorithms that outperform the choice of always selecting the default solver and outperform the featureless

approach (refer to Figures A.8 and A.9). We note that when the size of the portfolio is 14, which is the total number of solvers to choose from, the value of the performance is equal to the value of the virtual best performing portfolio selector (i.e., no selection method will be able to exceed this value).

## A.5.2 Primal-Dual Integrals as a Proxy for Time



**Figure A.9:** Portfolio performance on the M&C data set. Selectors were trained on Primal-Dual integral values and tested on their time performance.

One of the questions we sought to answer was whether or not the PDI values will act as a proxy for time information in training, since PDI values provide more information about the progress of the solver than does the timing data, especially in the case that many (or all) algorithms simply reach the time limit.

From Figure A.9 we show that in circumstances where time data is not available for training, it is still possible to meaningfully train using primal-dual integral data. The reader will note that this is not automatic – some of our predictor engines performed worse than the

featureless approaches, notably for portfolios of size one (attempting to predict only the best algorithm).

Since we are predicting not just a single algorithm but a portfolio of many algorithms, Figure A.9 also highlights an important point: competitive methods that work well for choosing a portfolio may not appear competitive when they are used to select only a single algorithm. This itself provides strong evidence that some of the previous research on single algorithm selection could behave very differently if revisited in the context of portfolio selection.

### A.5.3 Software package

We developed a Python software package named “Algorithm Predictor” to support our methodology. All results and plots in this work were produced with this package. Our package supports:

- **Feature Investigation** using various techniques (currently Pearson correlations, MDS and PCA) as well as machine learning methods (currently random regression forest). Each of these software modules produces plots and depending on the method, a list of features that it finds to be the most important.
- **Algorithm Selection** such as random regression forests, random forest classifiers and others combined with AdaBoost and Hydra boosting techniques. Currently, the Algorithm Predictor package supports training models based on input features and performance data for single or multiple seeds by splitting training data into 80:20 train to test ratio. Trained models can be serialized to a file, but using trained models for predicting on a single problem instance is not implemented yet.
- **Performance Measurement** for a single model or set of models using shifted geometric mean (SGM) explained in Section A.4.4. Additionally, these results can be plotted with variance and/or standard error overlaid.

- **Centralized Configuration System** that controls many parameters of our package.

We also produced documentation where the reader can find more details about our package, as well as tutorials for each point described above. To access the documentation, navigate to the *docs/build* directory and open *index.html* in any web browser. The package has been made publicly available under: <https://github.com/GregorCH/algoselection/>.

In addition to this package, the feature computation was implemented using SCIP as a callable library. We also created various Python scripts used for data cleaning and curation that we didn't include in our package because we found them very specific to our problem.

## A.6 Conclusion

Random forests seem to be the clear winner, which is a result consistent with the findings in [118]. We believe random forests somewhat get around this difficulty owing to their ability to locally segment off subspaces, and train on these spaces. Other methods seem to be more affected globally by various subspaces, which throws off the general predictive power of the machine learning method. Additionally, this is an indication of how homogeneous the M&C dataset actually is. We have demonstrated that accurate predictions can be made with our various methods and that our approaches outperform the featureless approach.

In each of our methods, we tune various parameters independently to minimize the testing error. In a future analysis, we would like to remove this assumption and scan the parameter space more effectively to find the global minimum of the testing error.

In the course of our experiments, we also identified a promising avenue for future research. In our experiments we used machine learning techniques and predictive engines to rank algorithms and produced portfolios based upon this ranking, in effect choosing the first best, second best, etc. in that order, demonstrating the usefulness of instance-specific features in this process. However, in the case that one is predicting a nontrivial portfolio (size more than one), it is possible to

improve upon the results by focusing on choosing portfolios that focus on including algorithms that complement the existing ones. It is the belief of the authors that one can use the features of instances to intelligently predict such a portfolio, but how to use the features to identify the correct portfolio has not yet been explored. We believe that research in this direction will be valuable in the application, for instance, by identifying the best set of algorithms to run in parallel.

## Acknowledgment

This research is in part a reprint of the material as it appears in ZIB-Report 18-17 [1]. The dissertation author was a primary investigator and author of this paper.

The work for this article has been partly conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF grant number 05M14ZAM) within the program “Graduate-Level Research in Industrial Projects for Students” (GRIPS) 2017. The described research activities have been partly funded by the Federal Ministry for Economic Affairs and Energy within the project BEAM-ME (ID: 03ET4023A-F). The authors would like to thank the Zuse Institute Berlin, the Institute for Pure and Applied Mathematics and the National Science Foundation for their contributions in resources and finances to this project. Special thanks to Daniel Hulme and Karsten Lehmann from Satalia for many fruitful discussions about the topic.

## A.7 Extracted Features

This section gives a detailed description of the features extracted from both the `M&C` and `Regions` datasets. For all instances where the value of some feature is undefined, we use a zero value as replacement. Further in this section, whenever we refer to nonzero or finite values, we do that in the context of the corresponding SCIP functions for checking if a value is zero or infinity.

More details can be found in the SCIP documentation<sup>7</sup>.

### A.7.1 Static features

Static features are extracted based on the problem instance objective function, the constraint matrix and the right hand side vector. It follows a list of static features based on the objective function of an instance:

- Logarithm of the number of all variables in the objective function.
- Percentage of all binary, integer, implicit integer, continuous, and fixed variables in the objective function, where fixed variables are variables with equal upper and lower bounds.
- Percentage of all nonzero binary, integer, implicit integer, continuous, and fixed variables in the objective function.
- Minimum, maximum, mean, standard deviation and median values (further called summary statistics) for each variable type (binary, integer, implicit integer, continuous, and fixed) and for all types together.
- Summary statistics of finite upper and lower bounds for all variables and amount of finite entries.
- Objective dynamism calculated as the logarithm of the ratio of the maximal and minimal element in the vector of all absolute nonzero objective coefficients.

Before feature extraction, every constraint (all coefficients from the constraint matrix, left and right hand sides) was normalized such that the maximal absolute coefficient value of is one. It follows a list of the features extracted from the constraint matrix, left and right hand sides of the problem instance:

---

<sup>7</sup><http://scip.zib.de/doc/html/>

- Logarithm of number of all constraints.
- Percentage of each linear constraint type in problem instance. All 16 constraint types used for classification are listed MIPLIB2010 webpage.<sup>8</sup>
- Summary statistic of finite right and left hand sides vectors (separately and together), including vectors density.
- Percentage of finite right and left hand sides.
- Summary statistics of number of nonzeros of each constraint.
- Summary statistics for vectors of mean, minimum, maximum and standard deviation values of linear constraint coefficients and also summary statistics for vector of ratio of maximum and minimum value for each linear constraint in problem instance.
- Constraint matrix density.
- Summary statistics for clique vector.
- Dynamism statistics which is calculated on vector of elements calculated for every linear constraint. Each value is calculated as logarithm of ratio of maximum and minimum nonzero coefficients for particular linear constraint.

### **A.7.2 Dynamic features**

In order to extract dynamic features, instances were given to the SCIP solver using the default settings, and the solving process was allowed to continue up until the completion of the solution at the root node. No data was extracted from the time after the branching process began. It is our belief that this partial solving process can be done efficiently for most instances since in practice most of the computational effort on hard instances is spent in branching.

---

<sup>8</sup><https://miplib.zib.de/>

More precisely, dynamic features used in this chapter are extracted from SCIP execution logs based on the SCIP statistics output which gives performance information, for instance, resolving, separating and LP statistics at the root node. More details about SCIP statistics output can be found in [115]. In the case that one of the features extracted in this manner has a value of zero for every instance in the dataset, we drop that feature for the purposes of our experiments. Also, we do not use information that is provided in units of time, since such information is highly machine dependent; however, it is conceivable that one could use this information in future study if one also took the standpoint that it is acceptable to demand that an end-user benchmark and train the predictor on the machine that will be later making the predictions. In our experiments, we have the expectation that the training and prediction will be completed on different machines, and so we do not incorporate dynamic time information into our feature set.

### **A.7.3 Top features for M&C**

The list of the important features obtained using the procedure described in Section A.3.4 for the M&C data set:

- Separators Cuts clique
- log constr with off presolve
- nvar all max with off presolve
- clique max with fast presolve
- log vars with off presolve
- Separators Calls strongcg
- clique max with default presolve
- rh constr ratio with default presolve
- RootNode FinalRootIters



## A.7.4 Top features for Regions

The list of the important features obtained using the procedure described in Section A.3.4 for the `Regions` data set:

`constr matrix density with default presolve`

`coeff bin mean with default presolve`

`coeff bin std with fast presolve`

`Separators Calls clique`

`Separators Calls impliedbounds`

`Presolvers Calls trivial`

`Presolvers Calls dualfix`

`coeff bin median with off presolve`

`coeff all median with off presolve`

## A.8 Completion Codes

- Ok
  - `'ok'`: a typical satisfactory completion code.
  - `'fail_solution_infeasible'`: the algorithm terminated and gave a solution, but that solution does not satisfy the constraints. This is usually due to rounding errors and other forms of numerical inaccuracy within the machine so can be considered `'ok'`.
  - `'solved_not_verified'`: the algorithm terminated and found a solution that was better than the reported bound in the instance definition (happens either because the instance had not previously been solved or because the bound was simply missing in the instance definition). This should mean that the algorithm has correctly solved

the instance, but because we do not have the ground truth we cannot confirm its correctness.

- Fail

- `'fail_abort'`: indicates a catastrophic failure of the algorithm resulting in its early termination
- `'fail_dual_bound'`: this only happened on one of our training instances. This happens when the algorithm times out but reports a lower bound on the dual which is inconsistent with the known lower bound.
- `'fail_objective_value'`: the algorithm terminated and gave a feasible solution, but that solution was not optimal.

- Timelimit

- `'timelimit'`: typical completion code when algorithm exceeds the time limit.
- `'better'`: will happen when the algorithm times out, but it did find at least one feasible solution that was better than the reported bound in the instance definition (happens either because the instance had not previously been solved or because the bound was simply missing in the instance definition)

# Appendix B

## Renormalization Group Flows and Morse Homology

In this chapter, we discuss general aspects of renormalization group flows in arbitrary dimensions using tools from Morse homology. More specifically, we propose an alternate version of the  $a$ -theorem, and in general the even dimensional  $c$ -theorem. We show that the strong condition on these theorems is unnecessary given a mathematical constraint we will define later, and apply our results to the  $a$ -theorem to show that the beta function can be written as a gradient flow.

### B.1 Introduction

Topological quantum field theories arise when one considers invariants of quantum field theories with no propagating physical states. For instance, Coleman [129] shows that in the case of Georgi-Glashow models, the existence of numerous connected components can be used to prove the existence of non-dissipative solutions, each labeled by a quantized quantity analogous to magnetic flux. Mathematically speaking, the zeroth homology has a consequence on what this flux is. In this example, studying even a zeroth homology resulted in a physically interesting

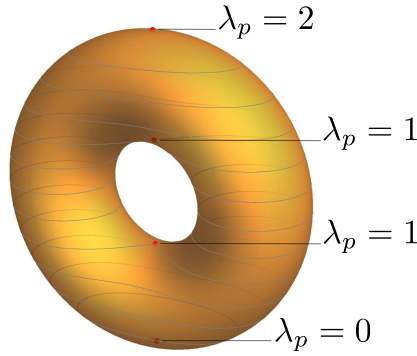
result.

In this chapter, rather than directly study the topological properties coming from fields living on manifolds, we investigate the topological properties coming from quantum field theory renormalization group (“RG”) flows living on manifolds. How these two distinct sets of properties are related is an interesting question which we unfortunately do not discuss here. The fixed points of these RG flows are conformal field theories (“CFTs”), and we are interested in the flows that originate from high energy (UV) points and end on low energy (IR) points. A recent study by Sergei Gukov [130] investigates topological properties of the RG space using tools from Morse homology by applying them to this set of flows. This novel method of studying the theory space structure may prove fruitful for a number of reasons, perhaps the most important being that it may allow for a deeper understanding of the  $c$ -theorem in any dimension.

This paper is roughly divided into two main sections: the first being a rough mathematical background of Morse homology, the second being an attempt to connect some of the results from this math to physically relevant quantities in RG flows. Not all of the results in the math section will find an application in our physics section, so for the hasty reader, we suggest skipping forward to the physics of it all and referring back when necessary.

## **B.2 Background**

In this chapter functions will be assumed to be smooth functions over a manifold  $M$ , unless specified otherwise, such that  $f : M \rightarrow \mathbb{R}$ . Intuitively,  $f$  can be thought of as a height function, so that level sets on  $M$  are given by  $f^{-1}(c)$ , where  $c$  is the “height”. Figure B.1 shows an example of the torus with level sets drawn in. For a more detailed treatment of the subject refer to [131, 132, 133, 134, 135, 136].



**Figure B.1:** Torus with level sets and critical points drawn in. With the appropriate coordinate system, the upmost points correspond to the largest heights.

## B.2.1 Classical Morse Theory

**Definition B.1** The **Hessian** is the matrix of second derivatives of the function evaluated at a point. Component-wise,  $H_{ij}(f, p) = \partial_i \partial_j f|_p$ . Generalizations of this exist for general smooth manifolds. A non-degenerate Hessian means we can definitively say what the curvature of the manifold is at every critical point, or equivalently, the determinant of the Hessian is non-zero.

**Definition B.2** The set of **critical points** of a continuous function  $f$  is defined to be the set

$$Cr(f) \equiv \{p \in M \mid df_p = 0\}$$

**Definition B.3** The function is said to be a **Morse function** if the Hessian at critical points is non-degenerate:

$$\forall p \in Cr(f) \Rightarrow |H_p(f)| \neq 0$$

**Lemma B.4 (Morse Lemma)** Let  $p$  be a critical point of a Morse function  $f$  on an  $n$ -dimensional manifold. Then locally,

$$f = f(p) - (y^1)^2 - (y^2)^2 - \dots - (y^{\lambda_p})^2 + (y^{\lambda_p+1})^2 + \dots + (y^n)^2 \quad (\text{B.1})$$

$\lambda_p$  is an invariant over the set of all Morse functions on  $M$ , so already there is some notion of topological information in the sense that these functions can be deformed into one another without changing this quantity.  $\lambda_p$  is called the **index** of the Morse function and is the number of negative eigenvalues of  $H_p(f)$  (i.e., the number of ways to independently descend the manifold at point  $p$ .) In Figure B.1 the topmost critical point has  $\lambda_p = 2$ ; the two critical interior points have  $\lambda_p = 1$ ; the bottommost critical point has  $\lambda_p = 0$ .

Since each critical point of a Morse function is non-degenerate, we can decompose the tangent space of a manifold into subspaces constructed by taking linear combinations of both the negative and positive eigenvectors of the Hessian independently:

$$\begin{aligned}
 T_p M &= T_p^s M \oplus T_p^u M, \text{ where:} \\
 T_p^s M &\equiv \text{span} \{v_{\lambda_i} \mid \lambda_i > 0\} \\
 T_p^u M &\equiv \text{span} \{v_{\lambda_i} \mid \lambda_i < 0\}
 \end{aligned}
 \tag{B.2}$$

Notice that  $\dim(T_p^u M) = \lambda_p$  and  $\dim(T_p^s M) = n - \lambda_p$

**Corollary B.5** *The set of non-degenerate critical points are isolated.*

**Corollary B.6** *A Morse function on a compact manifold has a finite number of critical points.*

Corollary B.6 will be useful when constructing a well defined homological boundary operator. In particular, it forces the coefficients of the terms in the boundary operator to be finite. If degeneracy is allowed, care has to be taken as these corollaries may not hold (we briefly mention this later).

## B.2.2 Morse-Smale Homology

Define a function  $\varphi$  to be a gradient flow of a Morse function which satisfies:

1.  $\varphi : \mathbb{R} \times M \rightarrow M$
2.  $\frac{\partial}{\partial t} \varphi(t, x) = -\nabla f(\varphi(t, x)) \equiv \tilde{\beta}$
3.  $\varphi(0, *) = id_M$

The second condition defines a vector field called  $\tilde{\beta}$ , which is given by minus the gradient of a Morse function called  $f$ . The second condition can be generalized to include “gradient-like” vector fields. If  $\beta$  is a gradient-like flow to the gradient flow  $\tilde{\beta}$ , the following conditions are met [137]:

- $\tilde{\beta} = \beta$ , in neighborhoods of critical points
- $\tilde{\beta}_i \beta^i > 0$ , away from critical points

Here, we assume there exists some invertible metric so that  $\chi^{ij} \tilde{\beta}_j = \tilde{\beta}^i$ . The first condition requires both vector fields to point in the same direction in neighborhoods of the critical points; the second condition requires the gradient-like vector field to point “roughly” in the same direction as the gradient field for all other points away from the critical points. Assuming these conditions on  $\beta$  are met, the structure of the flow lines given by  $\varphi$  can then be restated as:

1.  $\varphi : \mathbb{R} \times M \rightarrow M$
2.  $\frac{\partial}{\partial t} \varphi^i(t, x) = \beta^i$
3.  $\varphi(0, *) = id_M$

The results of Morse theory also hold with this looser structure on the vector field flows [133, 135].

**Definition B.7** *The **unstable** and **stable** manifolds of a manifold  $M$  are constructed by taking*

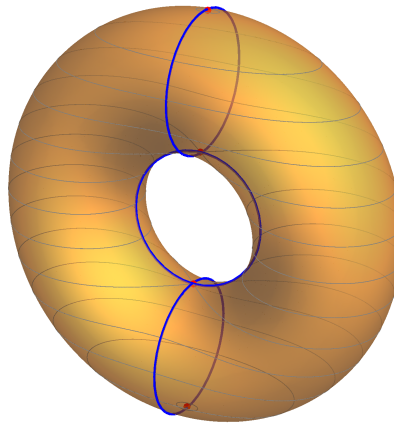
limits of  $\varphi$ :

$$W_p^u = \{x \in M \mid \lim_{t \rightarrow -\infty} \varphi(t, x) = p\} \quad (\text{B.3})$$

$$W_p^s = \{x \in M \mid \lim_{t \rightarrow +\infty} \varphi(t, x) = p\} \quad (\text{B.4})$$

**Corollary B.8**  $TW_p^u \cong T_p^u M \Rightarrow \dim(W_p^u) = \lambda_p$

It is useful to think of  $W_p^u$  as a  $\lambda_p$ -dimensional open ball since the stable and unstable manifolds give a CW-structure over  $M$ . Intuitively, the unstable manifold is the collection of flow lines flowing out of a critical point and the stable manifold is the collection of flow lines flowing in to a critical point. So, flows will start at unstable manifolds and flow to stable manifolds. Figure B.2 depicts this situation for a few example flows.



**Figure B.2:** Torus with level sets and a few examples of the stable and unstable manifolds. The topmost ring depicts just one possible flow coming from the  $\lambda = 2$  point, but there is indeed a two dimensional structure flowing from this point.

**Definition B.9** *If the stable and unstable manifolds of a Morse function all intersect transversally, the function is called **Morse-Smale**. Given any point  $x \in M$ , transversality means there exist flows going from critical points  $p \rightarrow q$  such that:*



$$T_x M = TW_p^u + TW_q^s$$

This means that at any point in  $M$ , the tangent spaces of the stable and unstable manifolds have enough information to construct the entire tangent space at that point. For example, the stable/unstable manifolds drawn for the height function in Figure B.2 don't satisfy the transversality condition between the two inner  $\lambda = 1$  critical points because  $TW_p^u = TW_q^s$  and each of these spaces are one-dimensional so the full tangent space cannot be constructed. However, a Morse function can always be made Morse-Smale by perturbing the metric; see Theorem B.11 for the general statement.

Morse-Smale functions are necessary to define a boundary operator because they flow out of unstable critical points to stable points of strictly lower indexes. This can be proven using Corollary B.10, which itself can be proven using the transversality condition of the stable and unstable manifolds and by a simple set theoretic argument.

**Corollary B.10** *For Morse-Smale functions and for flows starting at critical point  $p$  and ending on critical point  $q$ :*

$$\dim(W_p^u \cap W_q^s) = \lambda_p - \lambda_q \tag{B.5}$$

**Theorem B.11 (Kupka-Smale Theorem)** *For any metric on a manifold  $M$ , the set of smooth Morse-Smale functions is dense in the space of all smooth functions on  $M$ . Also, one can always find a Riemannian metric on  $M$  to make a Morse function into a Morse-Smale function.*

**Definition B.12** *For flows going from  $p \rightarrow q$  the **Moduli Space** is defined as:*

$$M(p, q) = (W_p^u \cap W_q^s) / \mathbb{R} \tag{B.6}$$

**Corollary B.13**  $\lambda_p - \lambda_q = 1 \Rightarrow M(p, q)$  *is a compact 0-dimensional manifold.*

A classification theorem states these manifolds are finite (i.e., there is a finite number of flows going from two critical points which differ in index by 1).

Following Gukov's work [130], the dimension of  $W_p^u$  can be thought of as the number of relevant operators at a critical point in the theory space of RG flows. Indeed, there are a certain number of relevant operators, each of which will flow from a  $UV \rightarrow IR$  critical point. This parallels the situation in Morse theory where the index of a critical point counts the number of independent ways there are to flow from a higher to lower index point.

Another result from Gukov's work follows directly from Corollary B.13. Namely, we should be able to count the number of flows between two points by considering the moduli space of the flows.

**Definition B.14** *The boundary operator can be defined as acting on the free Abelian group generated by critical points of a Morse-Smale function of index  $k$ :*

$$\begin{aligned} \partial_k : C_k(f) &\rightarrow C_{k-1}(f) \\ p &\rightarrow \sum_{q \in Cr_{k-1}(f)} \#M(p, q)q, \quad p \in Cr_k(f) \end{aligned} \tag{B.7}$$

$\#M(p, q) \in \mathbb{Z}$  is defined to be the signed number of flow lines from  $p \rightarrow q$ , and  $Cr_k(f)$  is the set of critical points of  $f$  of index  $k$ .

A necessary step for having this define a homology is to show that  $\partial^2 \equiv 0$ . We consider points  $p$  and  $q$  such that  $\lambda_p = i + 2$  and  $\lambda_q = i$ . Rather than go through the steps of the derivation, we hope to give an intuition (albeit a very tenuous one) for why this is true. We need to rely on a few results:

- $M(p, q)$  can be compactified to  $\overline{M(p, q)}$
- The boundary of  $\overline{M(p, q)}$  turns out to be equivalent to the action of  $\partial^2$ . In other words,  $\partial^2 p \sim \#\partial \overline{M(p, q)}$

- $\overline{M(p, q)}$  is an oriented 1-manifold with boundary, so its signed number of boundary points is zero

**Theorem B.15 (Morse Homology Theorem)** *The homology of the Morse-Smale chain complex  $(C_*(f), \partial_*)$  is isomorphic to the singular homology on  $M$ .*

**Corollary B.16**  $M = \coprod_p W_p^u = \coprod_p W_p^s$ . *i.e., the stable and unstable manifolds can be used to construct the original manifold  $M$ .*

### B.2.3 Extensions to Morse Homology

We briefly mention Morse-Bott and Floer homology, extensions of Morse-Smale homology each of which are best suited in certain circumstances. Both fields have a deep and beautiful structure which we unfortunately do not go into here.

In the case where the critical points are non-isolated, i.e., whereby  $Cr(f)$  now consists of manifolds rather than points, we should rely on Morse-Bott homology. Because the critical points are non-isolated the boundary operator is no longer well defined so the Morse Homology theorem does not hold. In this case, we should now set  $Cr(f) = \coprod_i U_i$  where  $U_i$  are connected submanifolds of  $M$  such that  $df|_{U_i} \equiv 0$ . If  $H_p(f)$  is non-degenerate in the normal direction for all  $p \in U_i$  then  $f$  is called **Morse-Bott**. By the Kupka-Smale theorem, a Morse-Bott function can be perturbed to a Morse-Smale function, which means the Morse homology theorem can be applied to this perturbed function! Refer to [138] for the explicit form of the perturbed function.

In the case where the critical points have infinite index, but still a finite relative index, we should rely on **Floer homology**. More specifically, Floer homology deals with the case where the Hessian operator decomposes into infinite dimensional spaces in which it is either positive or negative definite. The Yang-Mills Lagrangian appears in variants of Floer homology [139].

## B.3 The $c$ -theorems

### B.3.1 The $d = 2$ $c$ -theorem

There is an intuition that the RG flow is an irreversible process, where the number of massless degrees of freedom decreases along the flow: higher energies allow for more excited degrees of freedom to appear, and some of this information is lost as the flow can be seen as a course-graining procedure. In 2d field theory, Zamolodchikov's  $c$ -theorem [140] provides a function which satisfies this decreasing property and is equal to the central charge of the Virasoro algebra for CFTs. At fixed points, the 2d  $c$ -function is the trace anomaly that appears from regularizing the stress energy tensor:  $\langle T^\mu_\mu \rangle_{d=2} = -\frac{cR}{12}$  where  $R$  is the scalar curvature of the spacetime manifold. We refer to the total parameter space the flows live in as  $\mathcal{T}$  and the flows themselves as  $\mathcal{T}_{RG}$  (i.e.,  $\mathcal{T}_{RG} \subseteq \mathcal{T}$ ). Traditionally, an "RG time" is introduced as  $t = -\ln(\mu)$ , which increases toward the IR. With this variable, Zamolodchikov's theorem states:

**Theorem B.17 ( $c$ -theorem)** *There exists a positive-definite function  $c : \mathcal{T}_{RG} \rightarrow \mathbb{R}$  such that*

1. *Monotonically decreases along flow lines:*

$$\frac{dc(g)}{dt} = \beta^i(g) \frac{\partial c(g)}{\partial g^i} \leq 0$$

*where equality is met only at the fixed points of the RG flow*

2. *Stationary at fixed points:*

$$\beta^i(g_*) = 0 \implies \frac{\partial c(g)}{\partial g^i} \Big|_{g=g_*} = 0$$

3. *Equals the CFT central charge at fixed points*

These quantities have analogs to the objects in Morse homology, and we make these connections explicit in a later section. In Zamolodchikov's work, a positive definite symmetric

matrix appears that can be thought of as a metric over the theory space. It is used to relate the beta function to the derivative of the  $c$ -function with respect to the coupling constants:

$$\beta^i \frac{\partial c(g)}{\partial g^i} = -12\beta^i \beta^j G_{ij} \quad (\text{B.8})$$

### B.3.2 The even dimensional $c$ -theorem

For  $d \neq 2$  the stress-energy tensor trace takes on additional anomalies, so the proof of the theorem above currently only holds for  $d = 2$ . A possible generalization to even dimensions was proposed by Cardy [141], which collectively are now called  $c$ -theorems. The potential candidate for a higher dimensional  $c$ -function can be motivated using the generalized Gauss-Bonnet theorem which relates the geometry (in the sense of curvature) to the topology (in the sense of genus) of a spacetime manifold  $\mathcal{M}$ . For  $d = 2$ :

$$\int_{\mathcal{M}} \langle T_{\mu}^{\mu} \rangle_{d=2} \sqrt{g} d^2 x = -\frac{c}{12} \int_{\mathcal{M}} R \sqrt{g} d^2 x = -\frac{c}{12} \chi \quad (\text{B.9})$$

Here,  $\chi$  is the Euler characteristic of  $\mathcal{M}$ . In general, the Euler characteristic disappears for an odd dimensional compact manifold, so the theorem is only useful for gathering information for even  $d \geq 2$ . So as an ansatz, the arbitrary even dimensional  $c$ -function is taken to be:

$$C = \alpha_d \int_{\mathcal{M}} \langle T_{\mu}^{\mu} \rangle \sqrt{g} d^d x \quad (\text{B.10})$$

In the case of  $d = 4$ , the stress energy trace takes on an additional anomaly:  $\langle T_{\mu}^{\mu} \rangle_{d=4} = -aE_4 + cW^2$ .  $E_4$  is the 4-dimensional Euler density and  $W$  is the Weyl tensor. (Note that the Euler density is precisely the Ricci scalar in two dimensions). In this case, the candidate function is the constant in front of the Euler density. So for  $d = 4$  the  $c$ -theorem is known as the  $a$ -theorem, which has yet to be proven in full generality. A number of results have been proven about the  $a$ -theorem so far: Osborn has shown that the  $a$ -function decreases to all orders in perturbation

theory [142]; Fortin, Grinstein, and Stergiou showed RG flows which are cyclic correspond to CFTs which are consistent with the  $a$ -theorem (in this they showed that along the cycle,  $\frac{da}{dt} = 0$ , and that the whole cycle corresponds to a CFT) [143]; Komargodski and Schwimmer gave an argument in support of the theorem [144] in even dimensions. We discuss an application of Morse homology specifically to the case of  $d = 4$  in the last section. However, for the remainder of this section we focus on even dimensions.

Similar to the  $2d$   $c$ -theorem, there is a general even dimensional  $c$ -theorem (for instance, in [145]) which is constructed to satisfy similar requirements:

**Conjecture B.18 ( $c$ -theorem)** *For even dimensional QFTs there exists a positive-definite function  $c : \mathcal{T}_{RG} \rightarrow \mathbb{R}$  such that*

1. (*Weak Version*):  $c_{UV} > c_{IR}$
2. (*Medium Version*):  $\frac{dc(g)}{dt} \leq 0$  where equality is met at CFTs, and  $c$  is stationary at CFTs (i.e.,  $\frac{\partial c(g)}{\partial g^i} |_{g_*} = 0$ )
3. (*Strong Version*): The beta function is a gradient flow of the  $c$ -function with  $\beta^i(g) = G^{ij}(g) \frac{\partial c(g)}{\partial g^j}$ , with  $G_{ij} = (G^{ij})^{-1}$  and  $G^{ij}$  positive definite

We follow a similar delineation as Ken Intriligator and his collaborators have done [145] to distinguish the various levels of mathematical structure the  $c$ -function may satisfy (in their work, they specifically investigate the  $a$ -function). We call these various levels the “weak,” “medium,” and “strong” versions of the  $c$ -theorem. They are not independent claims due to the fact that if a stronger version of the  $c$ -theorem is satisfied, then all weaker conditions are automatically satisfied. We also point out that the medium version of the  $c$ -theorem in Intriligator’s work does not include the condition for the  $c$ -function to be stationary at fixed points. We include this condition as it is both reminiscent of Zamolodchikov’s work and because it will be necessary in a later proof.

### B.3.3 The proposed even dimensional $c$ -theorem

In this section, we introduce a new version of the even dimensional  $c$ -theorem by using tools from Morse homology. First, we make a few connections between the objects in this math and the objects that appear in the  $c$ -theorem:

#### Morse $c$ -Theorem

$$f : M \rightarrow \mathbb{R} \sim c : \mathcal{T}_{RG} \rightarrow \mathbb{R}$$

$$\varphi : \mathbb{R} \times M \rightarrow M \sim g : \mathbb{R} \times \mathcal{T}_{RG} \rightarrow \mathcal{T}_{RG}$$

$$\frac{\partial}{\partial t} \varphi^i(t, x) = -\nabla^i f(\varphi(t, x)) \sim \frac{\partial}{\partial t} g^i(t, g_0) = -\nabla^i c(g(t, g_0))$$

$$\varphi(0, *) = id_M \sim g(0, *) = id_{\mathcal{T}_{RG}}$$

Notice that the coupling constants are functions of the RG time and of the initial conditions of the flow. Also, recall that the mathematical tools from Morse theory can be applied to flows so long as they are gradient-like. So a natural question at this point is the following: does assuming the existence of a gradient-like flow on the RG space tell us anything about the  $c$ -function? A benefit to studying the RG space with algebraic topology, rather than the conventional differential geometry tools, is that topology is well suited to discovering global properties of the space which are invariant under certain deformations. This invariant information seems to be useful in uncovering properties of the  $c$ -function, as we have the following Conjecture B.19. We only partially prove the conjecture, and hence do not refer to it as a theorem. Conjecture B.19 can be proven as a theorem if one of the details (shown in bold) can be shown to be true.

**Conjecture B.19**  $\beta$  gradient-like  $\Leftrightarrow \frac{dc(g)}{dt} \leq 0$  where equality is met at CFTs, and  $c$  is stationary at CFTs

**Proof B.20** We denote an arbitrary beta function by  $\beta$ , and use  $\tilde{\beta}$  to denote a gradient flow (i.e.,  $\tilde{\beta} = -\nabla f$ ,  $f : \mathcal{T}_{RG} \rightarrow \mathbb{R}$ , where  $f$  is an arbitrary function). Recall, the two conditions for  $\beta$  to be

considered gradient-like to  $\tilde{\beta}$ , are that (1) these vector fields point in the same direction away from the critical points and (2) that they agree in neighborhoods of the critical points.

For the  $\Leftarrow$  implication:

$$\frac{dc}{dt} \leq 0 \Rightarrow -\frac{\partial c}{\partial g^i} \frac{dg^i}{dt} \geq 0 \Rightarrow \tilde{\beta}_i \beta^i \geq 0$$

where equality is met only at the fixed points as  $\beta^i(g_*) = 0$ . Additionally, the vector fields agree at the fixed points due to the fact that  $c$  is stationary at these points:  $\tilde{\beta}^i(g_*) = \beta^i(g_*) = 0$ . **However, we have yet to show that these fields agree in the neighborhoods, and hence this is not a full proof. We leave this detail for future work, or for the reader as an exercise.**

For the  $\Rightarrow$  implication:

$$\tilde{\beta}_i \beta^i \geq 0 \Rightarrow -\frac{\partial f}{\partial g^i} \frac{dg^i}{dt} \geq 0 \Rightarrow \frac{df}{dt} \leq 0$$

Additionally, this function will satisfy  $\frac{dg}{dt} = -\frac{\partial f}{\partial g}$  in neighborhoods of critical points as this is one of the conditions for being gradient-like. This implies that the function  $f$  is stationary at critical points. Recall that, according to Zamolodchikov's theorem, a stationary function is one such that  $\beta = 0 \Rightarrow \frac{\partial f}{\partial g} = 0$ .

We remark on a few subtleties in this proof and on some key points related to the proof:

- In the  $\Leftarrow$  direction, the gradient flow function  $f : \mathcal{T}_{RG} \rightarrow \mathbb{R}$  is automatically assumed to be the candidate  $c$ -function, and hence we have  $\beta$  being a gradient-like flow to  $\tilde{\beta} = -\nabla c$ . In the  $\Rightarrow$  direction, we show that there exists a function that monotonically decreases along the flows and is stationary at critical points, which are the conditions for the candidate  $c$ -function. It is just a matter of notation that we call this function  $f$ .
- The approach we took via the results from Morse theory allowed for a few line proof. If we instead used an actual  $c$ -function, the proof certainly would not have been so short or clean.



For instance, equation B.11 shows the  $a$ -function that Jack and Osborn derive [142], and it is not entirely clear how to prove Theorem B.19 in this case.

- Nothing in this discussion depends on the dimensionality of the spacetime. Although we introduce the application of Morse homology to renormalization group flows in the even dimensional case, there is no reason *a priori* these results cannot be applied to the odd dimensional case.

Stephen Smale proved that, given a gradient-like vector field, there exists a metric such that the field is the gradient of some function [134]. So  $\beta$  gradient-like  $\Rightarrow \beta^i = -\nabla^i f$  for some metric.

**Corollary B.21** *If the beta function defines a gradient-like flow, or equivalently if there exists a function that monotonically decreases along the flow and is stationary at fixed points, there exists a metric where the beta function is a gradient of some function.*

This suggests an alternate form of the  $c$ -theorem, with the strong version completely removed due to Corollary B.21 and with an equivalent form of the medium version rephrased in the context of Morse theory (i.e., the conditions 2a and 2b below are equivalent):

**Conjecture B.22 (alternate  $c$ -theorem)** *For even dimensional QFTs there exists a positive-definite function  $c : \mathcal{T}_{RG} \rightarrow \mathbb{R}$  such that*

1. (Weak Version):  $c_{UV} > c_{IR}$
- 2a. (Medium Version):  $\frac{dc(g)}{dt} \leq 0$  where equality is met at CFTs, and  $c$  is stationary at CFTs
- 2b. (Medium Version): The  $\beta$  function is gradient-like

### B.3.4 The $a$ -theorem

Corollary B.21 shows that for beta functions that define a gradient-like flow, or equivalently monotonically decrease, there is a metric which makes the beta function a gradient flow. In this section, we apply this result to the case of  $d = 4$  and use the notation  $\partial_i \equiv \frac{\partial}{\partial g^i}$ , and also  $g_*$  to denote critical points. Jack and Osborn derive a condition relating the  $a$ -function to the beta flow for  $d = 4$  [142]:

$$\begin{aligned} 8\partial_i a &= \beta^j [\chi_{ij} + (\partial_i W_j - \partial_j W_i)], \\ a &= \beta_b + \frac{1}{8} W_i \beta^i \end{aligned} \tag{B.11}$$

It is interesting to note that Zamolodchikov [140] derives a similar result perturbatively, the main difference being that the  $d = 2$  case is written as a gradient flow:

$$\beta^i = -\frac{1}{12} G^{ij} \partial_j c$$

In equation B.11, although different schemes change the form of the  $W$  tensors, the overall equation is covariant with respect to changing these schemes as the additional contributions cancel. If we rearrange and redefine some of the terms in this equation, and if we also assume that  $\chi^{ik} \chi_{kj} = \delta_j^i$ , we arrive at:

$$\begin{aligned} 8\chi^{ij} \partial_i a &= \beta^j + \chi^{ij} (\partial_i W_k - \partial_k W_i) \beta^k \\ \nabla_H^j a &= \beta^j + \zeta^j \end{aligned} \tag{B.12}$$

Here, we have made two substitutions. In the first place,  $\nabla_H^j a \equiv 8\chi^{ij} \partial_i a$ . This emphasizes the fact that we are taking the derivative with a specific choice in metric,  $H$ . With this choice, the beta function has an additional  $\zeta$  term. The second substitution we make is  $\chi^{ij} (\partial_i W_k - \partial_k W_i) \beta^k \equiv$

$\zeta^j$ . This emphasizes the fact that  $\zeta$  can be seen as a perturbation contribution. Thus, by using the  $H$  metric, we do not have a pure gradient flow. Using Conjecture B.22, we will be able to show that there is an appropriate modification to  $H$  which will eliminate the  $\zeta$  term. In order to use this Conjecture, we must show Equation B.12 satisfies gradient-like properties:

- $\chi^{ij}(\partial_i W_k - \partial_k W_i)\beta^k|_{g_*} \equiv 0$
- $\beta^2 > -\chi^{ij}(\partial_i W_k - \partial_k W_i)\beta^k \beta_j$

The first of these conditions is true at critical points since  $\beta^k(g_*) = 0$ . **Again, we leave out the proof that this condition is true in neighborhoods of  $g_*$ .** To verify the second condition, we rely on the explicit forms of  $\chi_{ij}$  and  $W_i$  which Jack and Osborn calculate using the usual perturbative loop expansion for gauge theories with a single coupling, multicomponent  $\phi^4$  scalar field theories, and field theories with Yukawa interactions. Up to the lowest order in each of these theories, the curl term vanishes due to  $W_i$  being linear in the couplings (or inverse proportional to a single coupling in the case of Yang-Mills), so the second requirement is also satisfied at this order.

More generally though, we can see that the second requirement is satisfied by counting powers of the couplings on each side of the inequality. Since each side has a factor of  $\beta$  with an upper and lower index, these can be removed from consideration. Thus, we determine how many factors of the coupling constants are contributed from  $\chi^{ij}(\partial_i W_k - \partial_k W_i)$ . Since we are working perturbatively, the inequality above will be true if  $\chi^{ij}(\partial_i W_k - \partial_k W_i) \sim g^n$  since  $1 > g^n \forall n > 0$ .

To leading order, we specify the dependence on the coupling for each of these objects for various physical theories in Table B.1. We point out that the calculations that Jack and Osborn perform are explicitly for  $\chi_{ij}$  and not  $\chi^{ij}$ , and that they don't consider theories with interactions between Yang-Mills and Dirac fields or Yang-Mills and  $\phi^4$  theories. The former case is dealt with simply by taking an inverse, and the latter case we handle by considering the leading order contributions in the Feynman diagrams. Each of the three theories have their own set of

coupling constants; however we do not distinguish between them in the table below as we are only interested in the leading order dependence of  $\chi^{ij}$  and  $W_i$  on the total number of factors of the couplings.

**Table B.1:** The leading order dependence of  $\chi$  and  $W$  on the couplings for various physical theories. We specify the loop-order to which each calculation was done in [142].

Theory	object, g-dependence, loop order		
	Yang-Mills with single coupling	$\chi^{ij}$	$g^2$
$W_i$		$g^{-1}$	2-loop
Scalar $\phi^4$	$\chi^{ij}$	1	3-loop
	$W_i$	$g^{+1}$	3-loop
Yukawa	$\chi^{ij}$	1	2-loop
	$W_i$	$g^{+1}$	2-loop
YM+ $\phi^4$	$\chi^{ij}$	$g^{+3}$	2-loop
$\phi^4$ +Dirac	$\chi^{ij}$	$g^{+1}$	3-loop
YM+Dirac	$\chi^{ij}$	0	2-loop

In the first three theories listed in the table, to leading order  $\chi^{ij}(\partial_i W_k - \partial_k W_i) \sim g^0$ . However the inequality above is still satisfied at this order because the curl term vanishes and additionally satisfied at higher orders because  $\chi^{ij}(\partial_i W_k - \partial_k W_i) \sim g^n$ . For the following two theories listed in the table,  $\chi^{ij}(\partial_i W_k - \partial_k W_i) \sim g^n$  to lowest order. And the last theory has a vanishing  $\chi^{ij}$  to lowest order, but has  $\chi^{ij}(\partial_i W_k - \partial_k W_i) \sim g^n$  at higher orders.

Since these conditions are met, we conclude that the beta function can be written as a gradient flow with respect to some metric other than  $\chi_{ij}$  in the case of  $d = 4$  quantum field theories.

## **B.4 Acknowledgements**

We'd like to thank Justin Roberts for the invaluable insights he provided into the world of Morse theory, and for putting up with us for so long.

# Appendix C

## Explanations of Select Algorithms

A GMM consists of fitting the data to a weighted sum of Gaussians:

$$p(x; \theta) = \sum_{c=1}^C \alpha_c \mathcal{N}(x; \mu_c, \Sigma_c) \quad (\text{C.1})$$

where  $\alpha_c$  is the weight of the component  $c$ ,  $0 < \alpha_c < 1$ ,  $\sum_{c=1}^C \alpha_c = 1$  and  $\theta$  is a list of parameters:

$$\theta = \{\alpha_1, \mu_1, \Sigma_1, \dots, \alpha_C, \mu_C, \Sigma_C\}$$

Here,  $\mu$  is the mean and  $\Sigma$  is the covariance matrix. The Gaussian probability density function  $\mathcal{N}$  is the standard bell curve in one-dimension. In a  $D$ -dimensional space, it is defined as:

$$\mathcal{N}(x; \mu, \Sigma) \equiv \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right] \quad (\text{C.2})$$

We can impose certain restrictions on the covariance matrix, which in turn will affect the performance of the GMM. We use four different forms of the covariance matrix in this chapter to see the affect of how these choices affect the performance of the GMM. These choices are *full*, *tied*, *diagonal*, and *spherical* (see Figure 4.9 to see how these choices affect the classification

accuracy):

- Full:  $\Sigma_c = \mathbb{E}[(x - \mu)(x - \mu)^T]_c$ , where each component receives its own covariance matrix
- Tied:  $\Sigma = \frac{1}{C} \sum_{c=1}^C \Sigma_c$ . So all components receive the same averaged covariance matrix.
- Diagonal:  $\Sigma_c = \text{diag}(\sigma_1, \dots, \sigma_D)_c$ , where each  $\sigma_i$  is the variance in the  $i$ -th coordinate
- Spherical:  $\Sigma_c = \text{diag}(\sigma_1, \dots, \sigma_D)_c$ , where all  $\sigma_i$  take on the same value

# Bibliography

- [1] Alexander Georges, Ambros Gleixner, Gorana Gojic, Robert Lion Gottwald, David Haley, Gregor Hendel, and Bartłomiej Matejczyk. Feature-based algorithm selection for mixed integer programming. Technical Report 18-17, ZIB, Takustr. 7, 14195 Berlin, 2018.
- [2] Gordon E Moore. Cramming more components onto integrated circuits, 1965.
- [3] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.
- [4] Jeff Leek. The elements of data analytic style. *J. Leek. Amazon Digital Services, Inc*, 2015.
- [5] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [7] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.
- [8] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.
- [9] Daniel Krefl and Rak-Kyeong Seong. Machine learning of calabi-yau volumes. *Physical Review D*, 96(6):066014, 2017.
- [10] Jonathan Carifio, James Halverson, Dmitri Krioukov, and Brent D Nelson. Machine learning in the string landscape. *Journal of High Energy Physics*, 2017(9):157, 2017.
- [11] Ryan Poplin, Avinash V Varadarajan, Katy Blumer, Yun Liu, Michael V McConnell, Greg S Corrado, Lily Peng, and Dale R Webster. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering*, 2(3):158, 2018.
- [12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.
- [13] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.



- [14] Understanding the bias-variance tradeoff. <http://scott.fortmann-roe.com/docs/BiasVariance.html>. (Accessed on 04/11/2019).
- [15] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.
- [16] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- [17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [18] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691*, 2017.
- [19] Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer, 1992.
- [20] Iñigo Barandiaran. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8), 1998.
- [21] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [22] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.
- [23] Ethics and data science — stanford data science initiative. <https://sdsi.stanford.edu/about/ethics-and-data-science>. (Accessed on 04/10/2019).
- [24] Data science ethics — coursera. <https://www.coursera.org/learn/data-science-ethics>. (Accessed on 04/10/2019).
- [25] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.
- [26] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence guarantees for rmsprop and adam in non-convex optimization and an empirical comparison to nesterov acceleration. 2018.

- [29] Karl Pearson F.R.S. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.
- [30] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.
- [31] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.
- [32] Cheng-Yuan Liou, Jau-Chi Huang, and Wen-Chie Yang. Modeling word perception using the elman network. *Neurocomputing*, 71(16-18):3150–3157, 2008.
- [33] Ingwer Borg and Patrick Groenen. Modern multidimensional scaling: Theory and applications. *Journal of Educational Measurement*, 40(3):277–280, 2003.
- [34] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
- [35] John C Gower and Gavin JS Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18(1):54–64, 1969.
- [36] Brian S Everitt. Finite mixture distributions. *Encyclopedia of statistics in behavioral science*, 2005.
- [37] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- [38] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les paralléloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.
- [39] David J Ketchen and Christopher L Shook. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal*, 17(6):441–458, 1996.
- [40] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Advances in neural information processing systems*, pages 281–288, 2004.
- [41] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.
- [42] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

- [43] Marcel van Gerven and Sander Bohte. *Artificial neural networks as models of neural information processing*. Frontiers Media SA, 2018.
- [44] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.
- [45] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [46] Der-Tsai Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE transactions on computers*, 100(6):478–487, 1982.
- [47] Allen Hatcher. *Algebraic topology*. , 2005.
- [48] Robert Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.
- [49] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.
- [50] Gunnar Carlsson and Vin De Silva. Topological approximation by small simplicial complexes, 2003.
- [51] Gurjeet Singh, Facundo Mémoli, and Gunnar E Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *SPBG*, pages 91–100, 2007.
- [52] Daniel Müllner and Aravindakshan Babu. Python mapper: An open-source toolchain for data exploration, analysis and visualization, 2013.
- [53] Gerd Gigerenzer and Henry Brighton. Homo heuristicus: Why biased minds make better inferences. *Topics in cognitive science*, 1(1):107–143, 2009.
- [54] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17*, pages 506–519, New York, NY, USA, 2017. ACM.
- [55] Gunnar Carlsson, Tigran Ishkhanov, Vin De Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International journal of computer vision*, 76(1):1–12, 2008.
- [56] Gurjeet Singh, Facundo Memoli, Tigran Ishkhanov, Guillermo Sapiro, Gunnar Carlsson, and Dario L Ringach. Topological analysis of population activity in visual cortex. *Journal of vision*, 8(8):11–11, 2008.

- [57] Jens Schmalzing and Krzysztof M Górski. Minkowski functionals used in the morphological analysis of cosmic microwave background anisotropy maps. *Monthly Notices of the Royal Astronomical Society*, 297(2):355–365, 1998.
- [58] Pratyush Pranav, Herbert Edelsbrunner, Rien Van de Weygaert, Gert Vegter, Michael Kerber, Bernard JT Jones, and Mathijs Wintraecken. The topology of the cosmic web in terms of persistent betti numbers. *Monthly Notices of the Royal Astronomical Society*, 465(4):4281–4310, 2016.
- [59] Gregory R Bowman, Xuhui Huang, Yuan Yao, Jian Sun, Gunnar Carlsson, Leonidas J Guibas, and Vijay S Pande. Structural insight into rna hairpin folding intermediates. *Journal of the American Chemical Society*, 130(30):9676–9678, 2008.
- [60] Pek Y Lum, Gurjeet Singh, Alan Lehman, Tigran Ishkanov, Mikael Vejdemo-Johansson, Muthu Alagappan, John Carlsson, and Gunnar Carlsson. Extracting insights from the shape of complex data using topology. *Scientific reports*, 3:srep01236, 2013.
- [61] Robert J Adler, Omer Bobrowski, and Shmuel Weinberger. Crackle: The persistent homology of noise. *arXiv preprint arXiv:1301.1466*, 2013.
- [62] Robert J Adler, Omer Bobrowski, Matthew S Borman, Eliran Subag, and Shmuel Weinberger. Persistent homology for random fields and complexes. In *Borrowing strength: theory powering applications—a Festschrift for Lawrence D. Brown*, pages 124–143. Institute of Mathematical Statistics, 2010.
- [63] Partha Niyogi, Stephen Smale, and Shmuel Weinberger. A topological view of unsupervised learning from noisy data. *SIAM Journal on Computing*, 40(3):646–663, 2011.
- [64] Anders Björner. Topological methods. *Handbook of combinatorics*, 2:1819–1872, 1995.
- [65] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.
- [66] Mathieu Carriere and Steve Oudot. Structure and stability of the one-dimensional mapper. *Foundations of Computational Mathematics*, 18(6):1333–1396, 2018.
- [67] Mickaël Buchet, Frédéric Chazal, Steve Y Oudot, and Donald R Sheehy. Efficient and robust persistent homology for measures. *Computational Geometry*, 58:70–96, 2016.
- [68] Leo Carlsson, Gunnar Carlsson, and Mikael Vejdemo-Johansson. Fibres of Failure: Classifying errors in predictive processes. *arXiv e-prints*, page arXiv:1803.00384, February 2018.
- [69] Chao Chen, Xiuyan Ni, Qinxun Bai, and Yusu Wang. A topological regularizer for classifiers via persistent homology. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2573–2582, 2019.

- [70] Qi Zhao and Yusu Wang. Learning metrics for persistence-based summaries and applications for graph classification. *arXiv preprint arXiv:1904.12189*, 2019.
- [71] Tamal K Dey, Jiayuan Wang, and Yusu Wang. Graph reconstruction by discrete morse theory. *arXiv preprint arXiv:1803.05093*, 2018.
- [72] P Pearson, D Muellner, and G Singh. Tdamapper: Analyze high-dimensional data using discrete morse theory(2015).
- [73] Hendrik Jacob van Veen and Nathaniel Saul. Keplermapper. <http://doi.org/10.5281/zenodo.1054444>, Jan 2019.
- [74] Henry Adams, Andrew Tausz, and Mikael Vejdemo-Johansson. Javaplex: A research software package for persistent (co) homology. In *International Congress on Mathematical Software*, pages 129–136. Springer, 2014.
- [75] Vidit Nanda. Perseus: the persistent homology software. *Software available at <http://www.sas.upenn.edu/~vnanda/perseus>*, 2012.
- [76] Dmitriy Morozov. Dionysus, a c++ library for computing persistent homology, 2007.
- [77] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed computation of persistent homology. In *2014 proceedings of the sixteenth workshop on algorithm engineering and experiments (ALENEX)*, pages 31–38. SIAM, 2014.
- [78] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, and Mariette Yvinec. The gudhi library: Simplicial complexes and persistent homology. In *International Congress on Mathematical Software*, pages 167–174. Springer, 2014.
- [79] Lek-Heng Lim. Hodge laplacians on graphs. *arXiv preprint arXiv:1507.05379*, 2015.
- [80] Mustafa Hajij, Bei Wang, and Paul Rosen. Mog: Mapper on graphs for relationship preserving clustering. *arXiv preprint arXiv:1804.11242*, 2018.
- [81] Raymond B Cattell. The scree test for the number of factors. *Multivariate behavioral research*, 1(2):245–276, 1966.
- [82] Svante Wold. Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4):397–405, 1978.
- [83] Patrick O Perry. Cross-validation for unsupervised learning. *arXiv preprint arXiv:0909.3052*, 2009.
- [84] Art B Owen and Patrick O Perry. Bi-cross-validation of the svd and the nonnegative matrix factorization. *The annals of applied statistics*, 3(2):564–594, 2009.
- [85] Peter D Hoff. Model averaging and dimension selection for the singular value decomposition. *Journal of the American Statistical Association*, 102(478):674–685, 2007.

- [86] Persi Diaconis, Sharad Goel, and Susan Holmes. Horseshoes in multidimensional scaling and local kernel methods. *The Annals of Applied Statistics*, pages 777–807, 2008.
- [87] Paul Breiding, Sara Kalisnik Verovsek, Bernd Sturmfels, and Madeleine Weinstein. Learning algebraic varieties from samples. *arXiv preprint arXiv:1802.09436*, 2018.
- [88] Matan Gavish and David L Donoho. The optimal hard threshold for singular values is  $4/\sqrt{3}$ . *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014.
- [89] Keith T Poole and Howard L Rosenthal. *Ideology and congress*, volume 1. Transaction Publishers, 2011.
- [90] Morris P Fiorina and Samuel J Abrams. Political polarization in the american public. *Annu. Rev. Polit. Sci.*, 11:563–588, 2008.
- [91] Geoffrey C Layman, Thomas M Carsey, and Juliana Menasce Horowitz. Party polarization in american politics: Characteristics, causes, and consequences. *Annu. Rev. Polit. Sci.*, 9:83–110, 2006.
- [92] Keith T Poole and Howard Rosenthal. Voteview. *University of California, San Diego. www.voteview.com. Poole, Keith T*, 2012.
- [93] Gerald C Wright and Brian F Schaffner. The influence of party: Evidence from the state legislatures. *American Political Science Review*, 96(2):367–379, 2002.
- [94] Vin De Silva and Gunnar E Carlsson. Topological estimation using witness complexes. *SPBG*, 4:157–166, 2004.
- [95] Gunnar Carlsson and Vin De Silva. Zigzag persistence. *Foundations of computational mathematics*, 10(4):367–405, 2010.
- [96] Andrey Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *Inst. Ital. Attuari, Giorn.*, 4:83–91, 1933.
- [97] Nickolay Smirnov. Table for estimating the goodness of fit of empirical distributions. *Annals of Mathematical Statistics*, 19(2):279–281, 1948.
- [98] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436444, 2015.
- [99] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.
- [100] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv e-prints*, page arXiv:1412.6572, December 2014.

- [101] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *International Conference on Learning Representations*, 2017.
- [102] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML'11*, pages 833–840, USA, 2011. Omnipress.
- [103] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- [104] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.
- [105] Silvia Biasotti, Bianca Falcidieno, and Michela Spagnuolo. Extended reeb graphs for surface understanding and description. In *International conference on discrete geometry for computer imagery*, pages 185–197. Springer, 2000.
- [106] Georges Reeb. On the singular points of a completely integrable form of pfaff or of a numerical function [on the singular points of a completely integrable pfaff form or of a numerical function]. *Accounts Rendus Acad. Sciences Paris*, 222:847 – 849, 1946.
- [107] John Milnor. *Morse Theory.(AM-51)*, volume 51. Princeton university press, 2016.
- [108] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.
- [109] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, volume 3, 2017.
- [110] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. Towards the first adversarially robust neural network model on MNIST. In *International Conference on Learning Representations*, 2019.
- [111] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 186–202, 2010.
- [112] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, pages 1542–1543, 2003.
- [113] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. ISAC-Instance-specific algorithm configuration. In *ECAI*, volume 215, pages 751–756, 2010.

- [114] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.
- [115] Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 4.0. Technical Report 17-12, ZIB, Takustr.7, 14195 Berlin, 2017.
- [116] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.
- [117] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming.
- [118] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.
- [119] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76. ACM, 2000.
- [120] Emilie Danna. Performance variability in mixed integer programming, 2008. Presentation at Workshop on Mixed Integer Programming.
- [121] Andrea Lodi and Andrea Tramontani. Performance variability in mixed-integer programming. *Tutorials in Operations Research*, pages 1–12, 2013.
- [122] Robert E. Bixby, Sebastião Ceria, Cassandra M. McZeal, and Martin W. P. Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58:12–15, 1998.
- [123] Tobias Achterberg, Thorsten Koch, and Alexander Martin. MIPLIB 2003. *Operations Research Letters*, 34(4):1–12, 2006.
- [124] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E. Bixby, Emilie Danna, Gerald Gamrath, Ambros M. Gleixner, Stefan Heinz, Andrea Lodi, Hans Mittelmann, Ted Ralphs, Domenico Salvagnin, Daniel E. Steffy, and Kati Wolter. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011.
- [125] Computational Optimization Research at Lehigh Laboratory. MIP instances. <https://coral.ise.lehigh.edu/data-sets/mixed-integer-instances/>. Visited 12/2017.



- [126] Timo Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.
- [127] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [128] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, 2007.
- [129] Sidney Coleman. *Aspects of symmetry: selected Erice lectures*. Cambridge University Press, 1988.
- [130] Sergei Gukov. Counting RG flows. *JHEP*, 01:020, 2016.
- [131] Matthias Schwarz. Morse homology. In *Progress in Mathematics*. Citeseer, 1993.
- [132] Yukio Matsumoto. *An introduction to Morse theory*, volume 208. American Mathematical Soc., 2002.
- [133] John M Franks. Morse-smale flows and homotopy theory. *Topology*, 18(3):199–215, 1979.
- [134] Stephen Smale. On gradient dynamical systems. *Annals of Mathematics*, pages 199–206, 1961.
- [135] John Willard Milnor, L Siebenmann, and J Sondow. *Lectures on the h-cobordism theorem*, volume 390. Princeton University Press Princeton, NJ, 1965.
- [136] Morris W Hirsch. *Differential topology*, volume 33. Springer Science; Business Media, 2012.
- [137] Mikhail Grinberg. Gradient-like flows and self-indexing in stratified morse theory. *Topology*, 44(1):175–202, 2005.
- [138] David E Hurtubise. Three approaches to morse-bott homology. *African Diaspora Journal of Mathematics. New Series*, 14(2):145–177, 2012.
- [139] Edward Witten. Monopoles and four-manifolds. *arXiv preprint hep-th/9411102*, 1994.
- [140] A. B. Zamolodchikov. Irreversibility of the Flux of the Renormalization Group in a 2D Field Theory. *JETP Lett.*, 43:730–732, 1986. [Pisma Zh. Eksp. Teor. Fiz.43,565(1986)].
- [141] John L. Cardy. Is There a c Theorem in Four-Dimensions? *Phys. Lett.*, B215:749–752, 1988.
- [142] H Osborn. Derivation of a four dimensional c-theorem for renormalisable quantum field theories. *Physics Letters B*, 222(1):97–102, 1989.
- [143] Jean-François Fortin, Benjamín Grinstein, and Andreas Stergiou. Limit cycles in four dimensions. *arXiv preprint arXiv:1206.2921*, 2012.

- [144] Zohar Komargodski and Adam Schwimmer. On renormalization group flows in four dimensions. *Journal of High Energy Physics*, 2011(12):1–20, 2011.
- [145] Edwin Barnes, Ken Intriligator, Brian Wecht, and Jason Wright. Evidence for the strongest version of the 4d a-theorem via a-maximization along rg flows. *Nuclear Physics B*, 702(1):131–162, 2004.