Advances in Machine Learning for Compositional Data

Elliott Gordon Rodríguez

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2022

Abstract

Advances in Machine Learning for Compositional Data

Elliott Gordon Rodríguez

Compositional data refers to simplex-valued data, or equivalently, nonnegative vectors whose totals are uninformative. This data modality is of relevance across several scientific domains. A classical example of compositional data is the chemical composition of geological samples, e.g., major-oxide concentrations. A more modern example arises from the microbial populations recorded using high-throughput genetic sequencing technologies, e.g., the gut microbiome. This dissertation presents a set of methodological and theoretical contributions that advance the state of the art in the analysis of compositional data. Our work can be divided along two categories: problems in which compositional data represents the input to a predictive model, and problems in which it represents the output of the model. For the first class of problems, we build on the popular log-ratio framework to develop an efficient learning algorithm for high-dimensional compositional data. Our algorithm runs orders of magnitude faster than competing alternatives, without sacrificing model quality. For the second class of problems, we define a novel exponential family of probability distributions supported on the simplex. This distribution enjoys attractive mathematical properties and provides a performant probability model for simplex-valued outcomes. Taken together, our results constitute a broad contribution to the toolkit of researchers and practitioners studying compositional data.

Table of Contents

Acknowledgments xi

Chapter 1: Introduction 1

1.1 Dissertation Outline 2

1.2 Publications 3

Chapter 2: Background 4

2.1 Motivating Examples 4

2.2 Log-Ratio Framework 5

2.3 Probability Distributions on the Simplex 7

Chapter 3: Learning Sparse Log-Ratios 9

3.1 Introduction 9

3.2 Background 12

3.2.1 Log-Ratio Analysis 12

3.2.2 Balances 13

3.2.3 Amalgamations 14

3.2.4 Other Related Work 15

3.3 Methods 16

3.3.1 Optimization Problem 16

3.3.2 Continuous Relaxation 17
3.3.3 Discretization 19
3.3.4 Regularization 20
3.3.5 CoDaCoRe Algorithm 20
3.3.6 Amalgamations 21
3.3.7 Extensions 21
3.4 Experiments 22
3.4.1 Results 23
3.4.2 Interpretability 25
3.4.3 Scaling to Liquid Biopsy Data 26
3.4.4 Simulation Study 28
3.5 Conclusion 28

Chapter 4: The Continuous Categorical 29
4.1 Introduction 29
4.2 Background 32
4.3 The Continuous Categorical Distribution 33
4.3.1 Convexity and Modes 34
4.3.2 Concentration of Mass 34
4.3.3 Exponential Family 36
4.3.4 Normalizing Constant 37
4.3.5 Mean and Variance 38
4.3.6 Related Distributions 39

4.4 Sampling . . . 39
4.5 Experiments . . . 42
4.5.1 Simulation Study . . . 42
4.5.2 UK Election Data . . . 43
4.5.3 Knowledge Distillation . . . 47
4.5.4 Latent Variable Models . . . 48
4.6 Conclusion . . . 49

Chapter 5: Uses and Abuses of the Cross Entropy Loss . . . 51
5.1 Introduction . . . 51
5.2 From the Cross-Entropy to the Continuous Categorical . . . 53
5.3 Continuous Categorical Label Smoothing . . . 55
5.3.1 Experiments . . . 56
5.4 Probabilistic Actor-Mimic Reinforcement Learning . . . 58
5.4.1 Experiments . . . 61
5.5 Discussion and Future Work . . . 63

Conclusion . . . 65

References . . . 67

Appendix A: Learning Sparse Log-Ratios . . . 79
A.1 Additional Methodological Detail . . . 79
A.1.1 Continuous Relaxation . . . 79
A.1.2 Discretization . . . 80

A.2 Datasets 81
A.3 Additional Experimental Detail 81
A.3.1 Critical Difference Diagrams 83
A.3.2 Ablations 84
A.3.3 Implementation Detail for Competing Methods 84
A.4 Simulation Study 86
A.4.1 Low-Dimensions 86
A.4.2 Heterogeneous balances 87
A.4.3 High-Dimensions 88
A.5 Software Package 97
A.5.1 Installation 97
A.5.2 Training the Model 98
A.5.3 Visualizing Results 100
A.5.4 Predicting on New Data 102
A.5.5 Controlling Overlap 104
A.5.6 Using Amalgamations 105
A.5.7 Continuous Outcomes 106
A.5.8 Tuning the Regularization 107
A.5.9 Covariate Adjustment 109
A.5.10 Incremental Fit 109
A.5.11 Joint Fit 112
A.5.12 Unsupervised Learning 114
A.5.13 Multi-Omics Integration 116

Appendix B: The Continuous Categorical 126
B.1 Derivation of the Normalizing Constant 126
B.2 Additional Properties of the CC Distribution 129
B.2.1 Mean and Covariance 129
B.2.2 KL Divergence 130
B.2.3 Moment Generating Function 130
B.2.4 Marginalization 130
B.3 Sampling 131
B.3.1 The 'Naive' Rejection Sampler 131
B.3.2 The Ordered Rejection Sampler 133
B.3.3 The Permutation Sampler 134
B.3.4 Performance 137

Appendix C: Uses and Abuses of the Cross Entropy Loss 142
C.1 Experimental details 142
C.1.1 Label Smoothing 142
C.1.2 Actor-Mimic Network 142

List of Figures

3.1 Gain in classification accuracy (relative to the "majority vote" baseline classifier) plotted against runtime. Each point represents one of 25 datasets, with size proportional to the input dimension. Note the x-axis is drawn on the log-scale. CoDaCoRe (with balances) is the only method that scales effectively to our larger datasets, while consistently achieving high predictive accuracy. Moreover, its performance is broadly consistent across smaller and larger datasets. 24

3.2 CoDaCoRe variable selection for the first (most explanatory) log-ratio on the Crohn disease data (Rivera-Pinto et al., 2018). For each of 10 independent bootstraps of the training set (80% of the data randomly sampled with stratification by case-control), we show which variables are selected in the numerator (blue) and denominator (orange) of the balance. CoDaCoRe learns remarkably consistent log-ratios across independent training sets. 25

4.1 Density heatmaps of the 2-dimensional CC (defined on the 3-simplex). We show a near-uniform example, followed by more extremal examples, as well as a bimodal example (with the modes necessarily at the extrema). Note that, while we have defined the CC distribution on the space $\{\mathbf{x} \geq \mathbf{0} : \sum_{i=1}^{K-1} x_i \leq 1\}$, we plot the density on the equivalent set $\{\mathbf{x} \geq \mathbf{0} : \sum_{i=1}^{K} x_i = 1\}$. 35

4.2 Empirical bias of the Dirichlet and CC MLEs as a function of sample size, for $K = 3$ (other values of K behaved similarly). The error bars show ± 1 standard deviation over different draws of the parameters of the ground-truth model from their respective priors. Regardless of whether the synthetic data is generated from a Dirichlet or a CC, only the CC estimator is unbiased. 43

4.3 Test error for the linear and MLP models with a Dirichlet and CC log-likelihood. The CC objective is better-behaved, training more smoothly and converging to the MLE, unlike the Dirichlet log-likelihood which diverges, causing overfitting. 44

4.4 Log-likelihood during training with different optimizers for the Dirichlet and the CC models. Consistently across optimizers, the CC objective converges to the MLE and the Dirichlet diverges. 46

4.5 Test error for different random initializations of the model parameters. The CC model converges, whereas the Dirichlet diverges along a highly variable path in the parameter space, and the models it finds in this path are suboptimal. 46

4.6 Test error for different regularization strengths for the Dirichlet models. Lighter shades of pink and orange indicate increasing regularization strengths. While sufficiently strong regularization does stabilize the Dirichlet, preventing it from diverging, it still does not outperform the (unregularized) CC. 48

4.7 Test error at different temperatures for the Dirichlet, XE and CC objectives. For each student model, this error measures the L_2 difference between its fitted values and those of the teacher model, on the test set. 50

5.1 Learned representations for the classes "airplane" (blue), "automobile" (orange), and "bird" (pink), projected to an informative 2-dimensional affine subspace spanning the template-vectors of the 3 classes. We show the same plot for samples from the training (above) and test set (below), for the unsmoothed baseline (left), LS (middle), and CC-LS (right), trained with regularization (following the top row of Table 5.1). Note that CC-LS does not concentrate clusters as tightly as LS, suggesting the potential for richer learned representations. 59

5.2 Training curves for the CC-AMN and AMN models, run on the simplified single-game objective. Solid lines reflect a moving average of the raw evaluation scores (faded lines). Each training epoch lasts 100 000 frames, with the evaluation scores being calculated from another 100 000 frames. While the actor-mimic models learn much faster than the DQN, the CC-AMN shows no improvement over the AMN model. 62

A.1 CoDaCoRe variable selection for the first (most explanatory) log-ratio on the Crohn disease data (Rivera-Pinto et al., 2018). For each of 10 independent bootstraps of the training set (80% of the data randomly sampled with stratification by case-control), we show which variables are selected in the numerator (blue) and denominator (orange) of the log-ratio. Both versions of CoDaCoRe, with balances (top) or amalgamations (bottom), learn remarkably consistent log-ratios across independent training sets. Differences between the sets selected by CoDaCoRe with balances vs. CoDaCoRe with amalgamations can be explained by differences in how the geometric mean vs. summation operations impact the log-ratio. The geometric mean, being more sensitive to small numbers, is more affected by the presence of rarer bacteria species like Dialister and Roseburia (as compared with the more common bacteria species like Haemophilus and Faecalibacterium). 82

A.2 Critical difference diagrams for 5 different performance metrics, computed using the Wilcoxon-Holm method at 5% significance. CoDaCoRe is the most computationally efficient and the sparsest (tied with selbal and pairwise log-ratios), as well as being tied to the leading position in all 3 metrics of predictive accuracy. 120

A.3 Pairwise win rates for each evaluation metric. The (i, j)th entry shows the proportion of runs in which the ith method achieved a better score than the jth method, where i indexes rows and j indexes columns. Note that, while CoDaCoRe (with default parameters) substantially outperforms in terms of runtime and sparsity, the only method that consistently outperforms in terms of predictive accuracy is CoDaCoRe with $\lambda = 0$. 121

A.4 Rank scores of CoDaCoRe (with default parameters) relative to its competitors regressed against 4 dataset-level metrics of interest. Each point represents a different dataset, of which there are 25 total. Trendlines were obtained from 4 separate univariate least squares fits. Class imbalance is shown as a number between 0.5 and 1, denoting the proportion of datapoints in the majority class. Signal-to-noise ratio is estimated using the out-of-sample accuracy of the Random Forest model (note this is a rough proxy intended for illustrative purposes only). Note that CoDaCoRe tends to outperform the most on datasets with a low number of samples, and balanced classes. 122
A.5 Variable selection performance for CoDaCoRe, selbal, and coda-lasso for a range of simulated datasets. True positive rates represent the proportion of active variables that were selected by the model, averaged over 10 independent draws of the synthetic dataset for each combination of k (active variables) and $\tilde{k}$ (inactive variables). False positive rates represent the proportion of inactive variables that were selected by the model, averaged over the same draws. In all, $3 \times 4 \times 10 = 120$ datasets were used to produce this table, and the response variables were simulated using balances. 123
A.6 Variable selection performance for CoDaCoRe, selbal, and coda-lasso for a range of simulated datasets. True positive rates represent the proportion of active variables that were selected by the model, averaged over 10 independent draws of the synthetic dataset for each combination of k (active variables) and p (input variables). False positive rates represent the proportion of inactive variables that were selected by the model, averaged over the same draws. In all, $5 \times 5 \times 10 = 250$ datasets were used to produce this table, and the response variables were simulated using balances. 124
A.7 Variable selection performance for CoDaCoRe and amalgam, for a range of simulated datasets. True positive rates represent the proportion of active variables that were selected by the model, averaged over 10 independent draws of the synthetic dataset for each combination of k (active variables) and p (input variables). False positive rates represent the proportion of inactive variables that were selected by the model, averaged over the same draws. In all, $5 \times 5 \times 10 = 250$ datasets were used to produce this table, and the response variables were simulated using summed log-ratios. 125

B.1 Shows the performance of 3 sampling algorithms across different dimensions K. Each histogram shows the distribution, over 100 trials, of the number of proposals required for 1 acceptance, on the log scale (base 10). The distributions are not exponential, since each of the 100 trials is sampled from a different $CC(\boldsymbol{\lambda})$ distribution, where the parameter follows independent and identically distributed $\boldsymbol{\lambda} \sim Dirichlet(1/K, \cdots, 1/K)$. Due to computational constraints, the number of proposals in each trial is right-censored, hence the large bars at the right end of the histograms. 138

List of Tables

3.1 Qualitative comparison of learning algorithms, ordered from most sparse (top) to least (bottom). CoDaCoRe is the only method that performs on all of our criteria. See Table 3.2 for a corresponding quantitative comparison. 16

3.2 Evaluation metrics shown for each method, averaged over 25 datasets × 20 random train/test splits. Standard errors are computed independently on each dataset, and then averaged over the 25 datasets. The models are ordered by sparsity, i.e., percentage of active input variables, which is given in the third column. CoDaCoRe (with balances) is the only learning algorithm that is simultaneously fast, sparse, and accurate. The bottom row shows the performance of Random Forest, a powerful black-box classifier which can be thought of as providing an approximate upper bound on the predictive accuracy of any interpretable model. 23

3.3 Evaluation metrics for the liquid biopsy data (Best et al., 2015), averaged over 20 independent 80/20 train/test splits. CoDaCoRe (with balances) achieves equal predictive accuracy as competing methods, but with much sparser solutions. Note that sparsity is expressed as an (integer) number of active variables in the model (not as a percentage of the total, as was done in Table 3.2). 27

4.1 Test errors and runtime for our regression models of the UK election data. Both in the linear and MLP case, the CC model beats the Dirichlet counterpart. 45

4.2 Test errors for the student models trained under the Dirichlet, XE, and CC objectives, as well as using the hard labels instead of the teacher model. We show the best values obtained over 5 random initializations and the 3 temperature settings of figure 4.7. 49

5.1 Ablation study for label smoothing on CIFAR-10. We show out-of-sample accuracy for our baseline classifier (w/o LS), as well as vanilla LS and CC-LS, both with $\varepsilon = 0.1$. Errors indicate the standard deviation over 10 random initializations of the network. We consider the effect of LS and CC-LS over the baseline under each combination of dropout, weight decay and batch normalization, and find that CC-LS provides significant outperformance in the absence of BatchNorm. 57

5.2 Ratio of within-cluster sum of squares over between-cluster sum of squares, for the learned representations of Figure 5.1. Each cell shows the mean ratio over 10 random initializations, with standard errors. 57

5.3 Mean evaluation score (and standard deviation) over the last 20 evaluation epochs (higher is better). With the exception of Pong, the performance of AMN and CC-AMN is similar. 62

A.1 Variable selection performance for CoDaCoRe, selbal, and coda-lasso for simulated datasets with two generative balances of heterogeneous effect sizes. In all, $4 \times 10 = 40$ datasets were used to produce this table. For CoDaCoRe and selbal, we can distinguish between the first and the second learned log-ratios, whereas coda-lasso learns a single log-linear combination for the entire dataset. 88
A.2 Data description. n denotes the number of observations, p the number of input variables. We also show the number of observations in the case and control groups. 90
A.3 Data description. 91
A.4 Average runtime over 20 train/test splits, in seconds. 92
A.5 Proportion of input variables active (%), averaged over 20 train/test splits 93
A.6 Out-of-sample accuracy (%) averaged over 20 train/test splits. 94
A.7 Out-of-sample AUC (%) averaged over 20 train/test splits. 95
A.8 Out-of-sample F1 score (%) averaged over 20 train/test splits. 96

Acknowledgements

First and foremost, I thank my advisor, John Cunningham, for his long-standing support. His guidance will be ever appreciated, both in the academic domain and beyond. I would also like to thank my committee members, professors John Paisley, Allison Lopatkin, Cindy Rush, Dave Blei, and Andrew Gelman, for helping to improve my work. I am grateful also for the many colleagues and collaborators with whom I had the chance to interact during my time at Columbia, among them Gabriel Loaiza Ganem, Thom Quinn, Wenda Zhou, Geoff Pleiss, Kelly Buchanan, Taiga Abe, Luhuan Wu, Dan Biderman, Andres Potapczynski, Yixin Wang, Sian Kitt, Jon Auerbach, and Florian Stebegg, and professors Guy Sella, Richard Zemel, and Liam Paninski.
A special thanks goes to the PhD students in my year: George Chu, Nick Galbraith, Nabarun Deb, Charles Margossian, Reed Palmer, Manuel Xu, and Shun Xu. Their company and spirit made our first year of qual prep genuinely enjoyable. I will miss the days and nights spent on the tenth floor lounge, with them and with the rest of our department colleagues. I will look forward to our continued reunions elsewhere.
Last, but not least, *quisiera dedicar esta tesis a mis padres y a mi hermano, a la familia Rodríguez Prieto, a los primos, y a los chavales del barrio.*

Chapter 1: Introduction

Parece que nunca aprendo
a administrarme los jurdeles.
Siempre vamos contramano
por las calles de Manhattan.

Los Delinqüentes (grabación inédita)

Compositional Data (CoDa) consist of vectors whose components are the proportions of some total. This total is typically uninformative, so that we can divide it out in order to express each component as relative to a fixed total of one unit. Equivalently, each compositional data point becomes an element of the simplex:

$$\Delta^K = \left\{ \boldsymbol{x} \in \mathbb{R}_+^K : \sum_{k=1}^{K} x_k = 1 \right\}. \tag{1.1}$$

In particular, a compositional dataset is simply a set of points in the simplex, $\mathcal{D} = \{\boldsymbol{x}_i \in \Delta^K\}_{i=1}^n$, and we will therefore also refer to CoDa as *simplex-valued data*, interchangeably.

As a result of the unit-sum constraint, statistical techniques designed for unconstrained real-valued data are problematic when applied to CoDa, an observation that was made as early as Pearson (1897). Consequently, bespoke treatment is necessary for the analysis of CoDa (Pawlowsky-Glahn, Egozcue, and Tolosana Delgado, 2007). However, it was not until the seminal work of Aitchison (1982) that CoDa became a discipline in its own right. Aitchison defined a Hilbert space structure on the simplex, known as the *Aitchison geometry*, and proposed the popular *log-ratio* framework, which we shall review in the next chapter. To the present day, these ideas underpin much of CoDa methodology.

Compositional data arises naturally in a multitude of applications across the natural and social

sciences. Historically, the development of CoDa techniques has been most closely associated to the geosciences, where the chemical composition of geological samples is an important object of study (Buccianti, Mateu-Figueras, and Pawlowsky-Glahn, 2006; Buccianti and Grunsky, 2014). More recently, with the advent of high-throughput genetic sequencing technology, CoDa methodology has enjoyed renewed interest in bioinformatics (Gloor et al., 2017; Quinn et al., 2018; Calle, 2019). This interest is particularly relevant to microbiology, where the composition of the human microbiome is known to associate with numerous health outcomes (Gilbert et al., 2018). However, techniques developed for geochemical CoDa typically do not scale to the much higher dimensional microbiome datasets, a problem we shall address in Chapter 3.

CoDa is also present across many other domains, such as ecology (Douma and Weedon, 2019), materials science (Na et al., 2014), economics (Fry, Fry, and McLaren, 2000), political science (Katz and King, 1999), and others. Notably, CoDa appears in a number of machine learning applications, including model compression (Buciluǎ, Caruana, and Niculescu-Mizil, 2006), knowledge distillation (Hinton, Vinyals, and Dean, 2015), actor-mimic learning (Parisotto, Ba, and Salakhutdinov, 2015), and label smoothing (Szegedy et al., 2016). These applications will be discussed in Chapters 4 and 5. In particular, there is a growing body of work at the intersection of ML and CoDa (Tolosana-Delgado et al., 2019; Quinn et al., 2020; Ostner, Carcy, and Müller, 2021; Coenders and Greenacre, 2021), to which this dissertation presents a direct contribution.

1.1 Dissertation Outline

This dissertation is organized into self-contained chapters that can be read separately. We devote Chapter 2 to a brief review of some background material that is not directly covered in the later chapters. While this material is of general relevance to the entirety of this dissertation, Section 2.2 is particularly relevant to Chapter 3, and Section 2.3 to Chapters 4 and 5.

At the highest level, our work can be divided into two categories: models where the input is CoDa (but the output is real-valued or discrete), and models where the output is CoDa (but the inputs are real-valued or categorical). The first category is studied in Chapter 3, where we develop

a novel learning algorithm for high-dimensional CoDa. We build our method into a software package (discussed in Appendix A.5), which we apply to a range of microbiome datasets in order to identify clinical biomarkers that are predictive and interpretable. The second category is studied in Chapters 4 and 5, where we introduce a novel exponential family of probability distributions supported on the simplex. Chapter 4 explores the theoretical properties of our novel distribution and demonstrates its performance as a likelihood model for simplex-valued outcomes. Note that our distribution is less performant as a probability distribution for latent variables in hierarchical models. Chapter 5 further explores the applications of our distribution in key machine learning tasks. As of yet, these applications are constrained to low-dimensional problems, due to numerical instability arising from the normalizing constant. Note also that, while this is not the main focus of our work, we briefly consider models where both the input and output are CoDa in Appendix A.5.13, as well as unsupervised learning for CoDa in Appendix A.5.12.

Last, note that there exist minor notational differences across chapters. For instance, in Chapter 4, the simplex is defined as a $K-1$ dimensional subset of $\mathbb{R}^{K-1}$, which has positive Lebesgue measure and facilitates the proof of the normalizing constant in Section B.1. On the other hand, in Chapter 5, we use the more common definition of the simplex as a $K-1$ dimensional subset of $\mathbb{R}^K$ with zero Lebesgue measure. At any rate, the relevant notation will be defined within each chapter or otherwise should be clear from context.

1.2 Publications

Chapter 3 was joint work with Thomas Quinn and John Cunningham, published as (Gordon-Rodriguez, Quinn, and Cunningham, 2022). This work also resulted in the `codacore` software package, publicly available on CRAN,[1] which is discussed in Appendix A.5. Chapter 4 was joint work with Gabriel Loaiza-Ganem and John Cunningham, published as (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020). Chapter 5 was joint work with Gabriel Loaiza-Ganem, Geoff Pleiss, and John Cunningham, published as (Gordon-Rodriguez et al., 2020).

[1] `https://cran.r-project.org/web/packages/codacore/index.html`. Python version available at `https://github.com/egr95/py-codacore`.

Chapter 2: Background

¿Qué le estará pasando al pobre Miguel,
que hace mucho tiempo que no sale?

Esperanza García "la del Maera"

2.1 Motivating Examples

We motivate the following Sections by briefly describing some applied examples of compositional data analysis:

- In a classical geochemical application, Aitchison (1984) studies the mineral composition of 25 rock specimens of coxite type. Each specimen represents a 5-dimensional vector over geochemical types whose components add up to 100%, i.e., a 5-part composition. The goal of the analysis is to describe the statistical variability of coxite compositions, and to determine whether the porosity of a coxite specimen depends on its composition.
- In a more recent application to metagenomic high-throughput sequencing data, (Gloor and Reid, 2016) examine the composition of the vaginal microbiome of women suffering from bacterial vaginosis. This is a higher-dimensional dataset where each component represents an aggregation of microbial species at the genus level. The goal of the analysis is to evaluate the association between the microbial composition and disease status.
- In the context of machine learning, the method of knowledge distillation (Hinton, Vinyals, and Dean, 2015) implicitly deals with compositional data regression. In particular, a small neural network is trained to predict the softmax outputs of a separate large neural network. These outputs are, by definition of the softmax, simplex-valued data. In contrast to the

previous examples, the goal of knowledge distillation is to obtain a small neural network that enjoys reduced computational cost at a minimal loss in accuracy.

2.2 Log-Ratio Framework

Aitchison (1982) proposed the *log-ratio* framework, whereby CoDa is mapped from the simplex onto Euclidean space, where the usual tools of statistical learning can then be applied. There are many possible log-ratio transformations, the simplest being the *additive log-ratio* (ALR) (Aitchison, 1982):

$$\mathrm{ALR}(\boldsymbol{x}) = \left[\log\left(\frac{x_1}{x_K}\right), \log\left(\frac{x_2}{x_K}\right), \ldots, \log\left(\frac{x_{K-1}}{x_K}\right)\right]. \tag{2.1}$$

In other words, each component is taken relative to a *reference*, in this case the last component, x_K, and the ratio is mapped onto real space by the natural logarithm. In practice, the ALR transformation is rarely used because any downstream analysis becomes sensitive to the choice of reference. A common workaround is the *centered log-ratio* (CLR) (Aitchison, 1982), which takes the geometric mean of all components as a reference:

$$\mathrm{CLR}(\boldsymbol{x}) = \left[\log\left(\frac{x_1}{g(\boldsymbol{x})}\right), \log\left(\frac{x_2}{g(\boldsymbol{x})}\right), \ldots, \log\left(\frac{x_{K-1}}{g(\boldsymbol{x})}\right)\right], \tag{2.2}$$

where $g(\boldsymbol{x}) = \left(\prod_{k=1}^{K} x_k\right)^{1/K}$. While the CLR saves the user from the choice of which component to use as a reference, the geometric mean entangles all the components of a composition in each CLR coordinate, hindering interpretability (Aitchison, 2008).[1] Moreover, while the individual output components, $[\mathrm{CLR}(\boldsymbol{x})]_k = \log\left(\frac{x_k}{g(\boldsymbol{x})}\right)$, are unconstrained real numbers, taken together the CLR vectors are still constrained to a $K-1$ dimensional subspace of $\mathbb{R}^K$, namely $\{\boldsymbol{v} \in \mathbb{R}^K : \sum_{k=1}^{K} v_k = 0\}$. The *isometric log-ratio* (ILR) aims to resolve these limitations by taking a set of log-ratio transformations called *balances* (Egozcue et al., 2003). A balance is defined as the

[1] Taking the geometric mean as a reference is also, in a sense, an arbitrary choice

log-ratio between geometric means:

$$B(\mathbf{x}) = \log\left(\frac{(\prod_{k\in K^+} x_k)^{\frac{1}{|K^+|}}}{(\prod_{k\in K^-} x_k)^{\frac{1}{|K^-|}}}\right), \tag{2.3}$$

where K^+ and K^- denote a pair of disjoint subsets of the indices $\{1,\ldots,K\}$. A full ILR transformation takes a total of $K-1$ balances according to a *sequential binary partition* of the input components. The ILR transformation enjoys appealing mathematical properties, in particular defining an isometry between the simplex and Euclidean space, however it has been criticized for adding unnecessary complexity (Aitchison, 2008). More importantly, choosing an optimal set of ILR coordinates is generally hard, requiring extensive domain knowledge or computationally expensive techniques that do not scale to high-dimensional CoDa (Pawlowsky-Glahn, Egozcue, and Tolosana Delgado, 2011; Martín-Fernández et al., 2018).

In most practical situations, however, a full ILR transformation is overkill. Rather, it is sufficient to identify a small number of "maximally important" balances for a given set of CoDa, much like one might represent real-valued data by means of a small number of principal components. For high-dimensional CoDa, it is also desirable that the balances used are sparse,[2] meaning that they take geometric means over a small number of input components. This sparsity facilitates the interpretability of any downstream analysis, as well as providing the additional benefit of regularization. Thus, the problem of selecting an ILR transformation is superseded by that of *balance selection*, which we shall discuss in depth in Chapter 3.

Balance selection is of great interest for classification problems with compositional inputs. For example, in many microbiome studies, balance selection can be thought of as identifying bacterial communities whose relative weights are associated with a disease. However, balances are also used in models with compositional outputs, or in the context of unsupervised learning (Pawlowsky-Glahn, Egozcue, and Tolosana Delgado, 2011; Martín-Fernández et al., 2018).

[2]Note that instead of parameterizing balances using the subsets K^+ and K^-, we can use a single K-dimensional vector $\mathbf{w}$ with kth entry equal to $+1$, -1, or 0 to denote membership of the kth input component to the numerator, denominator, or neither, respectively. Sparsity of the balance then corresponds to sparsity of the vector $\mathbf{w}$.

2.3 Probability Distributions on the Simplex

The transformation (via log-ratios) of CoDa from the simplex to unconstrained Euclidean space suggests the use of multivariable real-valued distributions downstream. The most classical choice is the multivariable normal; when applied to log-ratio-transformed data, this distribution is equivalent to the *logistic normal* (Atchison and Shen, 1980). For instance, modeling ALR-transformed data with a $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ distribution corresponds to the following logistic normal density on the simplex:

$$f(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{|(2\pi)^{K-1}\boldsymbol{\Sigma}|^{\frac{1}{2}}} \frac{1}{\prod_{k=1}^{K} x_k} e^{-\frac{1}{2}\left\{\log\left(\frac{\mathbf{x}_{-K}}{x_K}\right)-\boldsymbol{\mu}\right\}^{\top}\Sigma^{-1}\left\{\log\left(\frac{\mathbf{x}_{-K}}{x_K}\right)-\boldsymbol{\mu}\right\}}, \tag{2.4}$$

which can be seen by a standard change of variables. This distribution has been used in many classical CoDa applications (Aitchison, 1985; Mateu-Figueras, Barceló-Vidal, and Pawlowsky-Glahn, 1998; Buccianti, Mateu-Figueras, and Pawlowsky-Glahn, 2006).

More widely known, and equally relevant to the field of CoDa, is the Dirichlet distribution, defined on the simplex by the density:

$$f(\mathbf{x}; \boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{K} x_k^{\alpha_k - 1}. \tag{2.5}$$

The Dirichlet distribution arises as the conjugate prior to the multinomial distribution; it is therefore also of central importance in the context of latent variable models (Blei, Ng, and Jordan, 2003). Uses of the Dirichlet within CoDa methodology include *Dirichlet regression* (Campbell and Mosimann, 1987; Hijazi and Jernigan, 2009) and *Dirichlet Component Analysis* (Wang et al., 2008), amongst several others.

Several other probability distributions on the simplex have been proposed, both in the context of CoDa (Rayens and Srinivasan, 1994; Mateu-Figueras, Barceló-Vidal, and Pawlowsky-Glahn, 1998; Aitchison and Bacon-Shone, 1999), and beyond (Jang, Gu, and Poole, 2016; Stirn, Jebara, and Knowles, 2019; Potapczynski, Loaiza-Ganem, and Cunningham, 2020). Distributions on the

simplex will be discussed in more detail in Chapters 4 and 5.

Chapter 3: Learning Sparse Log-Ratios

¿Esa calle qué tendrá?
Siempre por la misma calle,
sabiendo de muchas más.

José Mercé

3.1 Introduction

The automatic discovery of sparse biomarkers that are associated with an outcome of interest is a central goal of bioinformatics. In the context of high-throughput sequencing (HTS) data, and *compositional data* (CoDa) more generally, an important class of biomarkers are the log-ratios between the input variables. However, identifying predictive log-ratio biomarkers from HTS data is a combinatorial optimization problem, which is computationally challenging. Existing methods are slow to run and scale poorly with the dimension of the input, which has limited their application to low- and moderate-dimensional metagenomic datasets.

Building on recent advances from the field of deep learning, we present *CoDaCoRe*, a novel learning algorithm that identifies sparse, interpretable, and predictive log-ratio biomarkers. Our algorithm exploits a *continuous relaxation* to approximate the underlying combinatorial optimization problem. This relaxation can then be optimized efficiently using the modern ML toolbox, in particular, gradient descent. As a result, CoDaCoRe runs several orders of magnitude faster than competing methods, all while achieving state-of-the-art performance in terms of predictive accuracy and sparsity. We verify the outperformance of CoDaCoRe across a wide range of microbiome, metabolite, and microRNA benchmark datasets, as well as a particularly high-dimensional dataset

that is outright computationally intractable for existing sparse log-ratio selection methods.[1]

High-throughput sequencing technologies have enabled the relative quantification of the different bacteria, metabolites, or genes, that are present in a biological sample. However, the nature of these recording technologies results in *sequencing biases* that complicate the analysis of HTS data. In particular, HTS data come as counts, whose totals are constrained to the capacity of the measuring device. These totals are an artifact of the measurement process, and do not depend on the subject being measured. Hence, HTS counts arguably should be interpreted in terms of *relative abundance*; thus, it follows that HTS data are an instance of compositional data (Gloor et al., 2016; Gloor et al., 2017; Quinn et al., 2018; Quinn et al., 2019; Calle, 2019).

Recall that, unlike unconstrained real-valued data, the compositional nature of CoDa results in each variable becoming negatively correlated to all others (increasing one component of a composition implies a relative decrease of the other components). It is well known that, as a result, the usual measures of association and feature attribution are problematic when applied to CoDa (Pearson, 1897; Filzmoser, Hron, and Reimann, 2009; Lovell et al., 2015). Consequently, bespoke methods are necessary for a valid statistical analysis (Gloor et al., 2017). Indeed, the application of CoDa methodology to HTS data, especially microbiome data, has become increasingly popular in recent years (Fernandes et al., 2013; Fernandes et al., 2014; Rivera-Pinto et al., 2018; Calle, 2019; Quinn, Gordon-Rodriguez, and Erb, 2021).

The standard approach for analyzing CoDa is based on applying *log-ratio* transformations to map our data onto unconstrained Euclidean space, where the usual tools of statistical learning apply (Pawlowsky-Glahn and Egozcue, 2006). The choice of the log-ratio transform offers the necessary property of scale invariance, but in the CoDa literature it holds primacy for a variety of other technical reasons, including *subcompositional coherence* (Aitchison, 1982; Pawlowsky-Glahn and Buccianti, 2011). Log-ratios can be taken over pairs of input variables (Aitchison, 1982; Greenacre, 2019b; Bates and Tibshirani, 2019) or aggregations thereof, typically geometric means (Aitchison, 1982; Egozcue et al., 2003; Egozcue and Pawlowsky-Glahn, 2005; Rivera-Pinto et al.,

[1]The CoDaCoRe package is available on CRAN and at https://github.com/egr95/R-codacore. Code and instructions for reproducing our results is available at https://github.com/cunningham-lab/codacore.

2018) or summations (Greenacre, 2019a; Greenacre, 2020; Quinn and Erb, 2020). The resulting features work well empirically, but also imply a clear interpretation: a log-ratio is a single composite score that expresses the overall quantity of one sub-population as compared with another. For example, in microbiome HTS data, the relative weights between sub-populations of related microorganisms are commonly used as clinical biomarkers (Rahat-Rozenbloom et al., 2014; Crovesy, Masterson, and Rosado, 2020; Magne et al., 2020). When the log-ratios are sparse, meaning they are taken over a small number of input variables, they define biomarkers that are particularly intuitive to understand, a key desiderata for predictive models that are of clinical relevance (Goodman and Flaxman, 2017).

Thus, learning sparse log-ratios is a central problem in CoDa. This problem is especially challenging in the context of HTS data, due to its high dimensionality (ranging from 100 to over 10,000 variables). Existing methods rely on stepwise search (Rivera-Pinto et al., 2018; Greenacre, 2019b) or evolutionary algorithms (Quinn and Erb, 2020; Prifti et al., 2020), which scale poorly with the dimension of the input. These algorithms are prohibitively slow for most HTS datasets, and thus there is a new demand for sparse and interpretable models that scale to high dimensions (Li, 2015; Cammarota et al., 2020; Susin et al., 2020).

This demand motivates the present Chapter, published as (Gordon-Rodriguez, Quinn, and Cunningham, 2022), in which we introduce CoDaCoRe, a novel learning algorithm for **Co**mpositional **Da**ta via **Co**ntinuous **Re**laxations. CoDaCoRe builds on recent advances from the deep learning literature on *continuous relaxations* of discrete latent variables (Jang, Gu, and Poole, 2016; Linderman et al., 2018); we design a novel relaxation that approximates a combinatorial optimization problem over the set of log-ratios. In turn, this approximation can be optimized efficiently using gradient descent, and subsequently discretized to produce a sparse log-ratio biomarker, thus dramatically reducing runtime without sacrificing interpretability nor predictive accuracy. The main contributions of our method can be summarized as follows:

- **Computational efficiency.** CoDaCoRe scales linearly with the dimension of the input. It runs several orders of magnitude faster than its competitors.

- **Interpretability.** CoDaCoRe identifies a set of log-ratios that are sparse, biologically meaningful, and ranked in order of importance. Our model is highly interpretable, and much sparser, relative to competing methods of similar accuracy and computational complexity.

- **Predictive accuracy.** CoDaCoRe achieves better out-of-sample accuracy than existing CoDa methods, and performs similarly to state-of-the-art black-box classifiers (which are neither sparse nor interpretable).

- **Ease of use.** We devise an adaptive learning rate scheme that enables CoDaCoRe to converge reliably, requiring no additional hyperparameter tuning.

3.2 Background

Our work focuses on the supervised learning problem $\mathbf{x}_i \mapsto y_i$, where the inputs $\mathbf{x}_i$ are HTS data (or any CoDa), and the outputs y_i are the outcome of interest. For many microbiome applications, $\mathbf{x}_i$ represents a vector of frequencies of the different species of bacteria that compose the microbiome of the ith subject. In other words, x_{ij} denotes the abundance of the jth species (of which there are p total) in the ith subject. The response y_i is often a binary variable indicating whether the ith subject belongs to the case or the control groups (e.g., sick vs. healthy). For HTS data, the input frequencies x_{ij} arise from an inexhaustive sampling procedure, so that the totals $\sum_{j=1}^{p} x_{ij}$ are arbitrary and the components should only be interpreted in relative terms (i.e., as CoDa) (Gloor and Reid, 2016; Gloor et al., 2017; Quinn et al., 2018; Calle, 2019). While many of our applications pertain to microbiome data, our method applies to any high-dimensional HTS data, including those produced by *Liquid Chromatography Mass Spectrometry* (Filzmoser and Walczak, 2014).

3.2.1 Log-Ratio Analysis

Our goal is to obtain sparse log-ratio transformed features that can be passed to a downstream classifier or regression function. As discussed, these log-ratios will result in interpretable features and scale-invariant models (that are also subcompositionally coherent), thus satisfying the key re-

quirements for valid statistical inference in the context of CoDa. The simplest such choice is the *pairwise log-ratio*, defined as $\log(x_{ij^+}/x_{ij^-})$, where j^+ and j^- denote the indexes of a pair of input variables (Aitchison, 1982). Note that the ratio cancels out any scaling factor applied to $\mathbf{x}_i$, preserving only the relative information, while the log transformation ensures the output is (unconstrained) real-valued. There are many such (j^+, j^-) pairs (to be precise, $p(p-1)/2 = O(p^2)$ of them). In order to select good pairwise log-ratios from a set of input variables, Greenacre (2019b) proposed a greedy step-wise search algorithm. This method produces a sparse and interpretable set of features, but it is prohibitively slow on high-dimensional datasets, as a result of the step-wise algorithm scaling quadratically in the dimension of the input. A heuristic search algorithm that is less accurate but computationally faster has been developed as part of Quinn et al. (2017), though its computational cost is still troublesome (as we shall see in Section 4.5). The *log-ratio lasso* is a computationally efficient alternative for selecting pairwise log-ratios (Bates and Tibshirani, 2019).

3.2.2 Balances

As discussed in Chapter 2, balances (Egozcue and Pawlowsky-Glahn, 2005) have become of interest in microbiome applications, due to their interpretability as the relative weight between two sub-populations of bacteria (Morton et al., 2019a; Quinn and Erb, 2019). Recall that balances are defined as the log-ratios between geometric means of two subsets of the input variables:[2]

$$\begin{aligned} B(\mathbf{x}_i; J^+, J^-) &= \log\left(\frac{(\prod_{j\in J^+} x_{ij})^{\frac{1}{p^+}}}{(\prod_{j\in J^-} x_{ij})^{\frac{1}{p^-}}}\right) \\ &= \frac{1}{p^+}\sum_{j\in J^+}\log x_{ij} - \frac{1}{p^-}\sum_{j\in J^-}\log x_{ij}, \end{aligned} \tag{3.1}$$

where J^+ and J^- denote a pair of disjoint subsets of the indices $\{1,\ldots,p\}$, and p^+ and p^- denote their respective sizes. For example, in microbiome data, J^+ and J^- are groups of bacteria species that may be related by their environmental niche (Morton et al., 2017) or genetic similarity

[2]Note that the original definition of balances includes a "normalization" constant, which we omit for clarity. This constant is in fact unnecessary, as it will get absorbed into a regression coefficient downstream. Note also that, in keeping with the notation of the present Chapter, we now use j, not k, to index the components of the input.

(Silverman et al., 2017; Washburne et al., 2017). Note that when $p^+ = p^- = 1$ (i.e., J^+ and J^- each contain a single element), $B(\mathbf{x}; J^+, J^-)$ reduces to a pairwise log-ratio. By allowing for the aggregation of more than one variable in the numerator and denominator of the log-ratio, balances provide a far richer set of features that allows for more flexible models than pairwise log-ratios. Insofar as the balances are taken over a small number of variables (i.e., J^+ and J^- are sparse), they also provide highly interpretable biomarkers.

The *selbal* algorithm (Rivera-Pinto et al., 2018) has gained popularity as a method for automatically identifying balances that predict a response variable. However, this algorithm is also based on a greedy step-wise search through the combinatorial space of subset pairs (J^+, J^-), which scales poorly in the dimension of the input and becomes prohibitively slow for many HTS datasets (Susin et al., 2020).

3.2.3 Amalgamations

An alternative to balances, known as *amalgamation*, is defined by aggregating components through summation:

$$A(\mathbf{x}_i; J^+, J^-) = \log\left(\frac{\sum_{j\in J^+} x_{ij}}{\sum_{j\in J^-} x_{ij}}\right), \tag{3.2}$$

where again J^+ and J^- denote disjoint subsets of the input components. Amalgamations have the advantage of reducing the dimensionality of the data through an operation, the sum, that some authors argue is more interpretable than a geometric mean (Greenacre, 2019a; Greenacre, Grunsky, and Bacon-Shone, 2020). On the other hand, amalgamations can be less effective than balances for identifying components that are statistically important, but small in magnitude, e.g., rare bacteria species (since small terms will have less impact on a summation than on a product).

Recently, Greenacre (2020) has advocated for the use of expert-driven amalgamations, using domain knowledge to construct the relevant features. On the other hand, Quinn and Erb (2020) proposed *amalgam*, an evolutionary algorithm to automatically identify amalgamated log-ratios

(Eq. 3.2) that are predictive of a response variable. However, this algorithm does not scale to high-dimensional data (albeit, comparing favorably to selbal), nor does it produce sparse models (hindering interpretability of the results). A similar evolutionary algorithm can be found in Prifti et al. (2020), however their model is not scale invariant, as is required by most authors in the field (Pawlowsky-Glahn and Egozcue, 2006).

3.2.4 Other Related Work

CoDa methodology has also recently attracted interest from the machine learning community (Tolosana-Delgado et al., 2019; Quinn et al., 2020; Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020; Gordon-Rodriguez et al., 2020; Templ, 2020). Relevant to us is *DeepCoDA* (Quinn et al., 2020), which combines self-explaining neural networks with log-ratio transformed features. In particular, DeepCoDA learns a set of *log-contrasts*, in which the numerator and denominator are defined as *unequally weighted* geometric averages of components (Egozcue and Pawlowsky-Glahn, 2016). As a result of this weighting, DeepCoDA loses much of the interpretability and intuitive appeal of balances (or amalgamations), which is exacerbated by its lack of sparsity. Moreover, like most deep architectures, DeepCoDA is sensitive to initialization and optimization hyperparameters (which limits its ease of use) and is susceptible to overfitting (which can further compromise interpretability of the model).

The special case of a linear log-contrast model has been referred to as *Coda-lasso*, and was separately proposed by Lu, Shi, and Li (2019). While Coda-lasso scales better than selbal, it has been found to perform worse in terms of predictive accuracy (Susin et al., 2020). More importantly, Coda-lasso is still prohibitively slow on the high-dimensional HTS data that we wish to consider. Last, we highlight another common set of features that are also a special case of log-contrasts: *centered-log-ratios* (CLR), where each input variable is divided by the geometric mean of *all* input variables (Aitchison, 1982). Models using these features, such as CLR-lasso (Susin et al., 2020), can be accurate and computationally efficient, however they are inherently not sparse and are difficult to interpret scientifically (Greenacre, 2019a).

Table 3.1: Qualitative comparison of learning algorithms, ordered from most sparse (top) to least (bottom). CoDaCoRe is the only method that performs on all of our criteria. See Table 3.2 for a corresponding quantitative comparison.

	Scalability	Interpretability	Sparsity	Accuracy
CoDaCoRe (ours)	+	+	+	+
Pairwise log-ratios (Greenacre, 2019b)	−	+	+	−
Selbal (Rivera-Pinto et al., 2018)	−	+	+	·
Lasso	+	+	+	−
Coda-lasso (Lu, Shi, and Li, 2019)	−	·	·	·
Amalgam (Quinn and Erb, 2020)	−	+	−	·
DeepCoDA (Quinn et al., 2020)	·	·	−	·
CLR-lasso (Susin et al., 2020)	+	−	−	+
Black-box (e.g., Random Forest, XGB)	+	−	−	+

3.3 Methods

We now present CoDaCoRe, a novel learning algorithm for HTS data, and more generally, high-dimensional CoDa. Unlike existing methods, CoDaCoRe is simultaneously scalable, interpretable, sparse, and accurate. We summarize the relative merits of CoDaCoRe and its competitors in Table 3.1.

3.3.1 Optimization Problem

In its basic formulation, CoDaCoRe learns a regression function of the form:

$$f(\mathbf{x}) = \alpha + \beta \cdot B(\mathbf{x}; J^+, J^-), \tag{3.3}$$

where B denotes a balance (Eq. 3.1), and α and β are scalar parameters. This regression function can be thought of in two stages: first we take the input and use it to compute a balance score, second we feed the balance score to a logistic regression classifier. For clarity, we will restrict our exposition to this formulation, but note that our algorithm can be applied equally to learn amalgamations instead of balances (see Section 3.3.6), as well as generalizing straightforwardly to nonlinear functions (provided they are suitably parameterized and differentiable).

Let $L(y, f)$ denote the cross-entropy loss, with $f \in \mathbb{R}$ given in logit space. The goal of CoDaCoRe is to find the balance that is maximally associated with the response. Mathematically, this can be written as:

$$\min_{(J^+, J^-, \alpha, \beta)} \sum_i L\left(y_i, \alpha + \beta \cdot B(\mathbf{x}_i; J^+, J^-)\right). \tag{3.4}$$

This objective function may look similar to a univariate logistic regression, however our problem is complicated by the joint optimization over the subsets J^+ and J^-, which determine the input variables that compose the balance. Note that the number of possible subsets of p variables is 2^p, so the set of possible balances is greater than 2^p and grows *exponentially* in p. Exact optimization is therefore computationally intractable for any but the smallest of datasets, and an approximate solution is required. Selbal corresponds to one such approximation, offering *quadratic* complexity in p, which is practical for low- to moderate-dimensional datasets ($p < 100$), but does not scale to high dimensions ($p >$ 1,000). As we shall now see, CoDaCoRe represents a critical improvement, achieving *linear* complexity in p which dramatically reduces runtime and enables, for the first time, the use of balances and amalgamations for the analysis of high-dimensional HTS data.

3.3.2 Continuous Relaxation

The key insight of CoDaCoRe is to approximate our combinatorial optimization problem (Eq. 3.4) with a continuous relaxation that can be trained efficiently by gradient descent. Our relaxation is inspired by recent advances in deep learning models with discrete latent variables (Jang, Gu, and Poole, 2016; Maddison, Mnih, and Teh, 2016; Linderman et al., 2018; Mena et al., 2018; Potapczynski, Loaiza-Ganem, and Cunningham, 2020). However, we are not aware of any similar proposals for optimizing over disjoint subsets, nor for learning balances or amalgamations in the context of CoDa.

Our relaxation is parameterized by an unconstrained vector of "assignment weights", $\mathbf{w} \in \mathbb{R}^p$, with one scalar parameter per input dimension (e.g., one weight per bacteria species). The weights

are mapped to a vector of "soft assignments" via:

$$\tilde{\mathbf{w}} = 2 \cdot \text{sigmoid}(\mathbf{w}) - 1 = \frac{2}{1 + \exp(-\mathbf{w})} - 1, \tag{3.5}$$

where the sigmoid is applied component-wise. Intuitively, large positive weights will max out the sigmoid, leading to soft assignments close to $+1$, whereas large negative weights will zero out the sigmoid, resulting in soft assignments close to -1. This mapping is akin to softly assigning input variables to the groups J^+ and J^-, respectively.

Let us write $\tilde{\mathbf{w}}^+ = \text{ReLU}(\tilde{\mathbf{w}})$ and $\tilde{\mathbf{w}}^- = \text{ReLU}(-\tilde{\mathbf{w}})$ for the (component-wise) positive and negative parts of $\tilde{\mathbf{w}}$, respectively. We approximate balances (Eq. 3.1) with the following relaxation:

$$\tilde{B}(\mathbf{x}_i; \mathbf{w}) = \frac{\sum_j \tilde{w}_j^+ \log x_{ij}}{\sum_j \tilde{w}_j^+} - \frac{\sum_j \tilde{w}_j^- \log x_{ij}}{\sum_j \tilde{w}_j^-} \tag{3.6}$$

$$= \frac{\tilde{\mathbf{w}}^+ \cdot \log \mathbf{x}_i}{\|\tilde{\mathbf{w}}^+\|_1} - \frac{\tilde{\mathbf{w}}^- \cdot \log \mathbf{x}_i}{\|\tilde{\mathbf{w}}^-\|_1}. \tag{3.7}$$

In other words, we approximate the geometric averages over subsets of the inputs, by *weighted* geometric averages over all components (compare Equations 3.1 and 3.6).

Crucially, this relaxation is differentiable in $\mathbf{w}$, allowing us to construct a surrogate objective function that can be optimized jointly in $(\mathbf{w}, \alpha, \beta)$ by gradient descent:

$$\min_{(\mathbf{w},\alpha,\beta)} \sum_i L\left(y_i, \alpha + \beta \cdot \tilde{B}(\mathbf{x}_i; \mathbf{w})\right). \tag{3.8}$$

Moreover, the computational cost of differentiating this objective function scales linearly in the dimension of $\mathbf{w}$, which overall results in linear scaling for our algorithm. We also note that the functional form of our relaxation (Eq. 3.6) can be exploited in order to select the learning rate adaptively (i.e., without tuning), resulting in robust convergence across all real and simulated datasets that we considered. We defer the details of our implementation of gradient descent to Appendix A.1.1.

3.3.3 Discretization

While a set of features in the form of Eq. 3.6 may perform accurate classification, a weighted geometric average over all input variables is much harder for a biologist to interpret (and less intuitively appealing) than a bona fide balance over a small number of variables. For this reason, CoDaCoRe implements a "discretization" procedure that exploits the information learned by the soft assignment vector $\tilde{\mathbf{w}}$, in order to efficiently identify a pair of sparse subsets, $\hat{J}^+$ and $\hat{J}^-$, which will define a balance.

The most straightforward way to convert the (soft) assignment $\tilde{\mathbf{w}}$ into a (hard) pair of subsets is by fixing a threshold $t \in (0, 1)$:

$$\tilde{J}^+ = \{j : \tilde{w}_j > t\}, \tag{3.9}$$

$$\tilde{J}^- = \{j : \tilde{w}_j < -t\}. \tag{3.10}$$

Note that given a trained $\tilde{\mathbf{w}}$ and a fixed threshold t, we can evaluate the quality of the corresponding balance $B(\mathbf{x}; \tilde{J}^+, \tilde{J}^-)$ (resp. amalgamation) by optimizing Eq. 3.4 over (α, β) alone, i.e., fitting a linear model. Computationally, fitting a linear model is much faster than optimizing Eq. 3.8, and can be done repeatedly for a range of values of t with little overhead. In CoDaCoRe, we combine this strategy with cross-validation in order to select the threshold, $\hat{t}$, that optimizes predictive performance (see Appendix A.1.2 for full detail). Finally, the trained regression function is:

$$\hat{f}(\mathbf{x}) = \hat{\alpha} + \hat{\beta} \cdot B(\mathbf{x}; \hat{J}^+, \hat{J}^-), \tag{3.11}$$

where $\hat{J}^+$ and $\hat{J}^-$ are the subsets corresponding to the optimal threshold $\hat{t}$, and $(\hat{\alpha}, \hat{\beta})$ are the coefficients obtained by regressing y_i against $B(\mathbf{x}_i; \hat{J}^+, \hat{J}^-)$ on the entire training set.

3.3.4 Regularization

Note from Equations 3.9 and 3.10 that larger values of t result in fewer input variables assigned to the balance $B(\mathbf{x}; \tilde{J}^+, \tilde{J}^-)$, i.e., a sparser model. Thus, CoDaCoRe can be regularized simply by making $\hat{t}$ larger. Similar to lasso regression, CoDaCoRe uses the *1-standard-error* rule: namely, to pick the sparsest model (i.e., the highest t) with mean cross-validated score within 1 standard error of the optimum (Friedman, Hastie, and Tibshirani, 2001). Trivially, this rule can be generalized to a λ-standard-error rule (to pick the sparsest model within λ standard errors of the optimum), where λ becomes a regularization hyperparameter that can be tuned by the practitioner if so desired (with lower values trading off some sparsity in exchange for predictive accuracy). In our public implementation, $\lambda = 1$ is our default value, and this is used throughout our experiments (except where we indicate otherwise). In practice, lower values (e.g., $\lambda = 0$) can be useful when the emphasis is on predictive accuracy rather than interpretability or sparsity, though our benchmarks showed competitive performance for any $\lambda \in [0, 1]$.

3.3.5 CoDaCoRe Algorithm

The computational efficiency of our continuous relaxation allows us to train multiple regressors of the form of Eq. 3.11 within a single model. In the full CoDaCoRe algorithm, we ensemble multiple such regressors in a stage-wise additive fashion, where each successive balance is fitted on the residual from the current model. Thus, CoDaCoRe identifies a *sequence* of balances, in decreasing order of importance, each of which is sparse and interpretable. Training terminates when an additional relaxation (Eq. 3.6) cannot improve the cross-validation score relative to the existing ensemble (equivalently, when we obtain $\hat{t} = 1$). Typically, only a small number of balances is required to capture the signal in the data, and as a result CoDaCoRe produces very sparse models overall, further enhancing interpretability. Our procedure is summarized in Algorithm 1.

Algorithm 1 CoDaCoRe

Inputs: Training data: $(\mathbf{x}_i, y_i)_{i=1}^n$.
Initialize $\hat{g}(\mathbf{x}) = 0$.
repeat
  Initialize a new relaxation $(\mathbf{w}, \alpha, \beta)$.
  Train $(\mathbf{w}, \alpha, \beta)$ by gradient descent.
  Use cross-validation to find the optimal threshold, $\hat{t}$.
  Retrain (α, β) using $(\hat{J}^+, \hat{J}^-)$.
  Update ensemble $\hat{g}(\mathbf{x}) \leftarrow \hat{g}(\mathbf{x}) + \hat{f}(\mathbf{x})$.
until $\hat{J}^+ = \emptyset$ or $\hat{J}^- = \emptyset$.
Return $\hat{g}(\mathbf{x})$.

3.3.6 Amalgamations

CoDaCoRe can be used to learn amalgamations (Eq. 3.2) much in the same way as for balances (the choice of which to use depending on the goals of the biologist). In this case, our relaxation is defined as:

$$\tilde{A}(\mathbf{x}_i; \mathbf{w}) = \log\left(\frac{\sum_j \tilde{w}_j^+ x_{ij}}{\sum_j \tilde{w}_j^- x_{ij}}\right) = \log\left(\frac{\tilde{\mathbf{w}}^+ \cdot \mathbf{x}_i}{\tilde{\mathbf{w}}^- \cdot \mathbf{x}_i}\right), \tag{3.12}$$

i.e., we approximate summations over subsets of the inputs, with *weighted* summations over all components (compare Eq. 3.2 and Eq. 3.12). The rest of the argument follows verbatim, replacing $B(\cdot)$ with $A(\cdot)$ and $\tilde{B}(\cdot)$ with $\tilde{A}(\cdot)$ in Equations 3.3, 3.4, 3.8, and 3.11.

3.3.7 Extensions

Our model allows for a number of extensions:

- *Unsupervised learning.* By means of a suitable unsupervised loss function, CoDaCoRe can be extended to unlabelled datasets, $\{\mathbf{x}_i\}_{i=1}^n$, as a method for identifying log-ratios that provide a useful low-dimensional representation. Such a method would automatically provide a scalable alternative to several existing dimensionality reduction techniques for CoDa (Pawlowsky-Glahn, Egozcue, and Tolosana Delgado, 2011; Mert, Filzmoser, and Hron, 2015; Martín-Fernández et al., 2018; Martino et al., 2019; Quinn, Gordon-Rodriguez, and

Erb, 2021). Such extensions have been explored by Quinn, Gordon-Rodriguez, and Erb (2021), and we discuss them briefly in Appendix A.5.12.

- *Incorporating confounders.* In addition to $(\mathbf{x}_i, y_i)_{i=1}^n$, in some applications the effect of additional (non-compositional) covariates, $\mathbf{z}_i$, is also of interest. In this case, the effect of $\mathbf{z}_i$ can be "partialled out" a priori by first regressing y_i on $\mathbf{z}_i$ alone, and using this regression as the initialization of the CoDaCoRe ensemble. Alternatively, $\mathbf{z}_i$ can also be modeled jointly in Equations 3.3 and 3.11 (e.g., by adding a linear term $\gamma \cdot \mathbf{z}_i$) (Forslund et al., 2015; Noguera-Julian et al., 2016; Rivera-Pinto et al., 2018). Both possibilities are briefly discussed in Appendix A.5.9.

- *Nonlinear regression functions.* Our method extends naturally to nonlinear regression functions of the form $f(\mathbf{x}) = h_\theta(B(\mathbf{x}; J^+, J^-))$, where h_θ is a parameterized differentiable family, including neural networks (Morton et al., 2019b; Quinn et al., 2020).

- *Applications to non-compositional data.* Aggregations of parts can be useful outside the realm of CoDa; for example, an amalgamation applied to a categorical variable with many levels represents a grouping of the categories (Bondell and Reich, 2009; Gertheiss and Tutz, 2010; Tutz and Gertheiss, 2016).

3.4 Experiments

We evaluate CoDaCoRe on a collection of 25 benchmark datasets including 13 datasets from the *Microbiome Learning Repo* (Vangay, Hillmann, and Knights, 2019), and 12 microbiome, metabolite, and microRNA datasets curated by Quinn and Erb (2019). These data vary in dimension from 48 to 3,090 input variables (see Appendix A.2 for a full description). For each dataset, we fit CoDaCoRe and competing methods on 20 random 80/20 train/test splits, sampled with stratification by case-control (He and Ma, 2013). Competing methods and their implementation are described in Appendix A.3.3.

Table 3.2: Evaluation metrics shown for each method, averaged over 25 datasets × 20 random train/test splits. Standard errors are computed independently on each dataset, and then averaged over the 25 datasets. The models are ordered by sparsity, i.e., percentage of active input variables, which is given in the third column. CoDaCoRe (with balances) is the only learning algorithm that is simultaneously fast, sparse, and accurate. The bottom row shows the performance of Random Forest, a powerful black-box classifier which can be thought of as providing an approximate upper bound on the predictive accuracy of any interpretable model.

	Runtime (s)	Vars. (%)	Acc. (%)	AUC (%)	F1 (%)
CoDaCoRe - Balances (ours)	**4.5±0.4**	**1.9±0.3**	75.2±2.4	79.5±2.6	73.7±2.6
CoDaCoRe - Amalgamations (ours)	**4.4±0.4**	1.9±0.3	71.8±2.4	74.5±2.8	69.8±2.9
selbal (Rivera-Pinto et al., 2018)	79,033.7±2,094.1	2.4±0.2	61.2±1.9	80.0±2.4	70.9±1.1
Pairwise Log-ratios (Greenacre, 2019b)	14,207.0±1,038.4	2.5±0.4	73.3±1.7	75.2±2.4	67.8±3.0
Lasso	**1.6±0.1**	4.4±0.6	72.4±1.7	75.2±2.3	65.2±3.7
CoDaCoRe - Balances w. $\lambda = 0$ (ours)	**9.8±2.2**	6.1±0.7	**77.6±2.2**	**82.0±2.3**	**76.0±2.5**
Coda-lasso (Lu, Shi, and Li, 2019)	1,043.0±55.4	19.7±2.7	72.5±2.3	78.0±2.4	64.2±4.4
amalgam (Quinn and Erb, 2020)	7,360.5±209.8	87.6±2.1	74.4±2.5	78.2±2.7	73.9±2.8
DeepCoDA (Quinn et al., 2020)	296.5±21.4	89.3±0.6	70.6±2.9	77.6±2.9	64.7±7.4
CLR-lasso (Susin et al., 2020)	**2.0±0.2**	100.0±0.0	77.5±1.8	81.6±2.2	75.8±2.7
Random Forest	10.6±0.4	·	78.0±2.2	82.2±2.2	77.3±2.5

3.4.1 Results

We evaluate the quality of our models across the following criteria: computational efficiency (as measured by runtime), sparsity (as measured by the percentage of input variables that are active in the model), and predictive accuracy (as measured by out-of-sample accuracy, ROC AUC, and F1 score). Table 3.2 provides an aggregated summary of the results; CoDaCoRe (with balances) is performant on all metrics. Indeed, our method provides the only interpretable model that is simultaneously scalable, sparse, and accurate. Detailed performance metrics on each of the 25 datasets are provided in Appendix A.3, together with critical difference diagrams for each of our success metrics.

Figure 3.1 shows the average runtime of our classifiers on each dataset, with larger points denoting larger datasets. On these common benchmark datasets, CoDaCoRe trains up to 5 orders of magnitude faster than existing interpretable CoDa methods. On our larger datasets (3,090 inputs), selbal runs in ∼100 hours, pairwise log-ratios and amalgam both run in ∼10 hours, and CoDaCoRe runs in under 10 seconds (full runtimes are provided in Table A.4). All runs, includ-

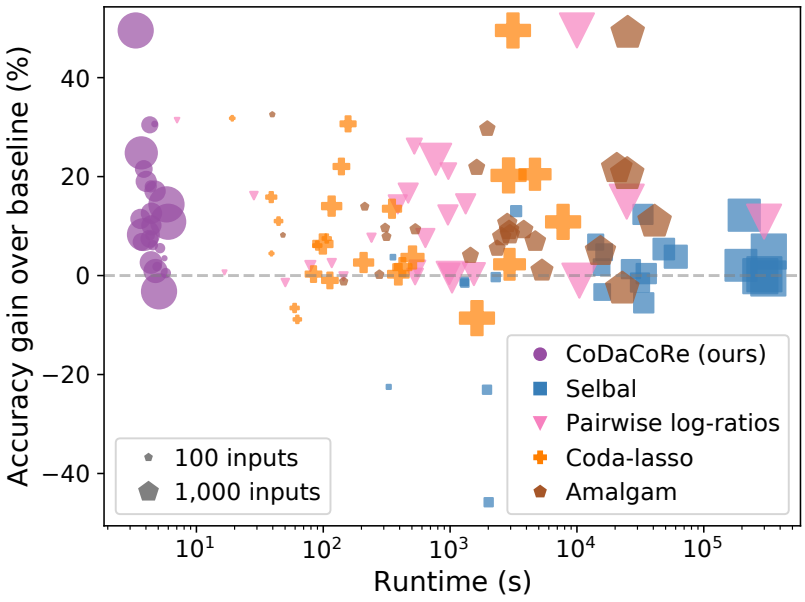


Figure 3.1: Gain in classification accuracy (relative to the "majority vote" baseline classifier) plotted against runtime. Each point represents one of 25 datasets, with size proportional to the input dimension. Note the x-axis is drawn on the log-scale. CoDaCoRe (with balances) is the only method that scales effectively to our larger datasets, while consistently achieving high predictive accuracy. Moreover, its performance is broadly consistent across smaller and larger datasets.

ing those involving gradient descent, were performed on identical CPU cores; CoDaCoRe can be accelerated further using GPUs, but we did not find it necessary to do so. It is also worth noting that the outperformance of CoDaCoRe is not merely as a result of the other methods failing on high-dimensional datasets. Section A.3.2 and Figure A.4 in the Appendix show that CoDaCoRe performs consistently across low- and high-dimensional datasets, and enjoys better sample efficiency than competing methods. Better sample efficiency could represent a particular advantage in biomedical studies, where most datasets have low n and high p.

Not only is CoDaCoRe sparser and more accurate than other interpretable models, it also performs on par with state-of-the-art black-box classifiers. By simply reducing the regularization parameter, from $\lambda = 1$ to $\lambda = 0$, CoDaCoRe (with balances) achieved an average 77.6% out-of-sample accuracy of and 82.0% AUC, on par with Random Forest and XGBoost (bottom rows of

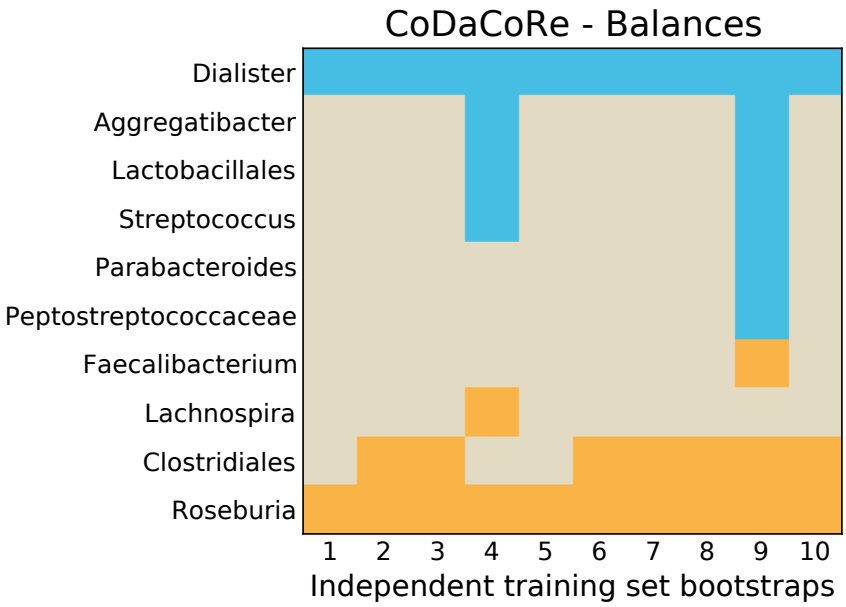


Figure 3.2: CoDaCoRe variable selection for the first (most explanatory) log-ratio on the Crohn disease data (Rivera-Pinto et al., 2018). For each of 10 independent bootstraps of the training set (80% of the data randomly sampled with stratification by case-control), we show which variables are selected in the numerator (blue) and denominator (orange) of the balance. CoDaCoRe learns remarkably consistent log-ratios across independent training sets.

Table 3.2), while only using 5.9% of the input variables, on average. This result indicates, first, that CoDaCoRe provides a highly effective algorithm for variable selection in high-dimensional HTS data. Second, the fact that CoDaCoRe achieves similar predictive accuracy as state-of-the-art black-box classifiers, suggests that our model may have captured a near-complete representation of the signal in the data. At any rate, we take this as evidence that log-ratio transformed features are indeed of biological importance in the context of HTS data, corroborating previous microbiome research (Rahat-Rozenbloom et al., 2014; Crovesy, Masterson, and Rosado, 2020; Magne et al., 2020).

3.4.2 Interpretability

The CoDaCoRe algorithm offers two kinds of interpretability. First, it provides the analyst with sets of input variables whose aggregated ratio predicts the outcome of interest. These sets are easy

to understand because they are discrete, with each component making an equivalent (unweighted) contribution. They are also sparse, usually containing fewer than 10 features per ratio, and can be made sparser by adjusting the regularization parameter λ. Such ratios have a precedent in microbiome research, for example the Firmicutes-to-Bacteroidetes ratio is used as a biomarker of gut health (Crovesy, Masterson, and Rosado, 2020; Magne et al., 2020). Second, CoDaCoRe ranks predictive ratios hierarchically. Due to the ensembling procedure, the first ratio learned is the most predictive, the second ratio predicts the residual from the first, and so forth. Like principal components, the balances (or amalgamations) learned by CoDaCoRe are naturally ordered in terms of their explanatory power. This ordering aids interpretability by decomposing a multivariable model into comprehensible "chunks" of information.

Notably, we find a high degree of stability in the log-ratios selected by the model. We repeated CoDaCoRe on 10 independent training set splits of the Crohn disease data provided by Rivera-Pinto et al. (2018), and found consensus among the learned models. Figure 3.2 shows which bacteria were included for each split. Importantly, the bacteria that were selected consistently by CoDaCoRe – notably Dialister, Roseburia and Clostridiales – were also identified by Rivera-Pinto et al. (2018). In Appendix A.3, we also present a comparison of Figure 3.2 when using CoDaCoRe to learn amalgamations instead of balances. The amalgamations tend to select more abundant bacteria species like Faecalibacterium rather than rarer species like Roseburia (due to the geometric mean being more sensitive to small numbers than the summation operator).

3.4.3 Scaling to Liquid Biopsy Data

HTS data generated from from clinical blood samples can be described as a "liquid biopsy" that can be used for cancer diagnosis and surveillance (Best et al., 2015; Alix-Panabières and Pantel, 2016). These data can be very high-dimensional, especially when they include all gene transcripts as input variables. In a clinical context, the use of log-ratio predictors is an attractive option because they automatically correct for inter-sample sequencing biases that might otherwise limit the generalizability of the models (Dillies et al., 2013). Unfortunately, existing log-ratio

Table 3.3: Evaluation metrics for the liquid biopsy data (Best et al., 2015), averaged over 20 independent 80/20 train/test splits. CoDaCoRe (with balances) achieves equal predictive accuracy as competing methods, but with much sparser solutions. Note that sparsity is expressed as an (integer) number of active variables in the model (not as a percentage of the total, as was done in Table 3.2).

	Runtime (s)	Vars (#)	Acc. (%)	AUC (%)	F1 (%)
CoDaCoRe	31±2.2	**3±1**	**91.0±1.9**	93.6±2.6	**94.4±1.2**
Lasso	23±0.2	22±4	87.8±1.3	94.7±1.5	92.7±0.7
RF	383±8.6	·	89.0±1.6	94.1±1.8	93.1±1.0
XGBoost	108±1.6	·	90.6±1.9	**95.9±1.5**	94.1±1.1

methods like selbal and amalgam simply cannot scale to liquid biopsy data sets that contain as many as 50,000 or more input variables.

The large dimensionality of such data has restricted its analysis to overly simplistic linear models, black-box models that are scalable but not interpretable, or suboptimal hybrid approaches where input variables must be pre-selected based on univariate measures (Best et al., 2015; Zhang et al., 2017; Sheng, Dong, and Xie, 2018). Owing to its linear scaling, CoDaCoRe can be fitted to these data at a similar computational cost to a single lasso regression, i.e., under a minute on a single CPU core. Thus, CoDaCoRe can be used to discover interpretable and predictive log-ratios that are suitable for liquid biopsy cancer diagnostics, among other similar applications.

We showcase the capabilities of CoDaCoRe in this high-dimensional setting, by applying our algorithm to the liquid biopsy data of (Best et al., 2015). These data contain p = 58,037 genes sequenced in n = 288 human subjects, 60 of whom were healthy controls, the others having been previously diagnosed with cancer. Averaging over 20 random 80/20 train/test splits of this dataset, we found that CoDaCoRe achieved the same predictive accuracy as competing methods (within error), but obtained a much sparser model. Remarkably, CoDaCoRe identified log-ratios involving just 3 genes, that were equally predictive to both black-box classifiers and linear models with over 20 active variables. This case study again illustrates the potential of CoDaCoRe to derive novel biological insights, and also to develop learning algorithms for cancer diagnosis, a domain in which model interpretability – including sparsity – is of paramount importance (Wan et al., 2017).

3.4.4 Simulation Study

In addition to the above previous experiments, we provide an extensive simulation study in Appendix A.4. For simulated HTS datasets of dimensionality ranging from 100 to 10,000 input variables, we find that CoDaCoRe is able to recover the true biomarkers used in the data-generating process, and does so with similar or higher accuracy (and orders of magnitude faster) than its competitors.

3.5 Conclusion

Our results corroborate the summary in Table 3.1: CoDaCoRe is the first sparse and interpretable CoDa model that can scale to high-dimensional HTS data. It does so convincingly, with linear scaling that results in runtimes similar to linear models. Our method is also competitive in terms of predictive accuracy, performing comparably to powerful black-box classifiers, but with interpretability. Our findings suggest that CoDaCoRe could play a significant role in the future analysis of high-throughput sequencing data, with broad implications in microbiology, statistical genetics, and the field of CoDa.

Chapter 4: The Continuous Categorical

Ya estoy saliendo con otra,
la recojo a media tarde
y no se va de mi vera,
se muere por ser mi novia
mi amiga la borrachera.

Paco Candela

4.1 Introduction

Simplex-valued data appear throughout statistics and machine learning, for example in the context of transfer learning and compression of deep networks. Existing models for this class of data rely on the Dirichlet distribution or other related loss functions; here we show these standard choices suffer systematically from a number of limitations, including bias and numerical issues that frustrate the use of flexible network models upstream of these distributions. We resolve these limitations by introducing a novel exponential family of distributions for modeling simplex-valued data – the *continuous categorical*, which arises as a nontrivial multivariate generalization of the recently discovered continuous Bernoulli. Unlike the Dirichlet and other typical choices, the continuous categorical results in a well-behaved probabilistic loss function that produces unbiased estimators, while preserving the mathematical simplicity of the Dirichlet. As well as exploring its theoretical properties, we introduce sampling methods for this distribution that are amenable to the reparameterization trick, and evaluate their performance. Lastly, we demonstrate that the continuous categorical outperforms standard choices empirically, across a simulation study, an applied

example on multi-party elections, and a neural network compression task.[1]

The importance of studying probability distributions on the simplex is motivated by the numerous applications of compositional data across the natural and social sciences (see Pawlowsky-Glahn and Egozcue (2006), Pawlowsky-Glahn and Buccianti (2011), and Pawlowsky-Glahn, Egozcue, and Tolosana-Delgado (2015) for an overview). Prominent examples appear in highly cited work ranging from geology (Pawlowsky-Glahn and Olea, 2004; Buccianti, Mateu-Figueras, and Pawlowsky-Glahn, 2006), chemistry (Buccianti and Pawlowsky-Glahn, 2005), microbiology (Gloor et al., 2017), genetics (Quinn et al., 2018), psychiatry (Gueorguieva, Rosenheck, and Zelterman, 2008), ecology (Douma and Weedon, 2019), environmental science (Filzmoser, Hron, and Reimann, 2009), materials science (Na et al., 2014), political science (Katz and King, 1999), public policy (Breunig and Busemeyer, 2012), economics (Fry, Fry, and McLaren, 2000), and the list goes on. An application of particular interest in machine learning arises in the context of model compression, where the class probabilities outputted by a large model are used as 'soft targets' to train a small neural network (Buciluǎ, Caruana, and Niculescu-Mizil, 2006; Ba and Caruana, 2014; Hinton, Vinyals, and Dean, 2015), an idea used also in transfer learning (Tzeng et al., 2015; Parisotto, Ba, and Salakhutdinov, 2015).

The existing statistical models with compositional outputs come in three flavors. Firstly, there are models based on a Dirichlet likelihood, for example the *Dirichlet GLM* (Campbell and Mosimann, 1987; Gueorguieva, Rosenheck, and Zelterman, 2008; Hijazi and Jernigan, 2009) and *Dirichlet Component Analysis* (Wang et al., 2008; Masoudimansour and Bouguila, 2017). Secondly, there are also models based on applying classical statistical techniques to log-ratio transformed data (Aitchison, 1982; Aitchison, 1994; Aitchison, 1999; Egozcue et al., 2003; Ma et al., 2016; Quinn et al., 2020). Thirdly, the machine learning literature has proposed predictive models that forgo the use of a probabilistic objective altogether and optimize the categorical cross-entropy instead (Ba and Caruana, 2014; Hinton, Vinyals, and Dean, 2015; Sadowski and Baldi, 2018). The first type of model, fundamentally, suffers from an ill-behaved loss function (amongst other draw-

[1] Our code is available at `https://github.com/cunningham-lab/cb_and_cc`

backs) (§5.2), which constrain the practitioner to using inflexible (linear) models with only a small number of predictors. The second approach typically results in complicated likelihoods, which are hard to analyze and do not form an exponential family, forgoing many of the attractive theoretical properties of the Dirichlet. The third approach suffers from ignoring normalizing constants, sacrificing the properties of maximum likelihood estimation (see Loaiza-Ganem and Cunningham (2019) for a more detailed discussion of the pitfalls).

In this chapter, which was published as (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020), we resolve these limitations simultaneously by defining a novel exponential family supported on the simplex, the continuous categorical (CC) distribution. Our distribution arises naturally as a multivariate generalization of the recently discovered continuous Bernoulli (CB) distribution (Loaiza-Ganem and Cunningham, 2019), a $[0, 1]$-supported exponential family motivated by Variational Autoencoders (Kingma and Welling, 2014), which showed empirical improvements over the beta distribution for modeling data which lies close to the extrema of the unit interval. Similarly, the CC will provide three crucial advantages over the Dirichlet and other competitors: it defines a coherent, well-behaved and computationally tractable log-likelihood (§4.3.1, 4.3.4), which does not blow up in the presence of zero-valued observations (§4.3.2), and which produces unbiased estimators (§4.3.3). Moreover, the CC model presents no added complexity relative to its competitors; it has one fewer parameter than the Dirichlet and a normalizing constant that can be written in closed form using elementary functions alone (§4.3.4). The continuous categorical brings probabilistic machine learning to the analysis of compositional data, opening up avenues for future applied and theoretical research.

It is important to note that our experiments focus on the advantages of the continuous categorical *as a likelihood model for compositional data*. On the other hand, simplex-valued random variables are also commonplace in the context of hierarchical models such as *Latent Dirichlet Allocation* (Blei, Ng, and Jordan, 2003). For this class of models, our novel distribution is generally not advantageous relative to standard choices, as discussed in Section 4.5.4.

4.2 Background

Careful consideration of the Dirichlet clarifies its shortcomings and the need for the CC family. The Dirichlet distribution is parameterized by $\boldsymbol{\alpha} \in \mathbb{R}_+^K$ and defined on the simplex $\mathbb{S}^{K-1} = \{\mathbf{x} \in \mathbb{R}_+^{K-1} : \sum_{i=1}^{K-1} x_i < 1\}$ by:

$$p(x_1, \ldots, x_{K-1}; \boldsymbol{\alpha}) = \frac{1}{\mathcal{B}(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}, \tag{4.1}$$

where $\mathcal{B}(\boldsymbol{\alpha})$ denotes the multivariate beta function, and $x_K = 1 - x_1 - \cdots - x_{K-1}$.[2]

This density function presents an attractive combination of mathematical simplicity, computational tractability, and the flexibility to model both interior modes and sparsity, as well as defining an exponential family of distributions that provides a multivariate generalization of the beta distribution and a conjugate prior to the multinomial distribution. As such, the Dirichlet is by far the best known and most widely used probability distribution for modeling simplex-valued random variables (Ng, Tian, and Tang, 2011). This includes both the latent variables of mixed membership models (Erosheva, 2002; Blei, Ng, and Jordan, 2003; Barnard et al., 2003), and the statistical modeling of compositional outcomes (Aitchison, 1982; Campbell and Mosimann, 1987; Hijazi, 2003; Wang et al., 2008). Our work focuses on the latter, with special attention to the setting where we aim to learn a (possibly nonlinear) regression function that models a simplex-valued response in terms of a (possibly large) set of predictors. In this context, and in spite of its strengths, the Dirichlet distribution suffers from three fundamental limitations.

First, flexibility: while the ability to capture interior modes might seem intuitively appealing, in practice it thwarts the optimization of predictive models of compositional data (as we will demonstrate empirically in section §4.5.2). To illustrate why, consider fitting a Dirichlet distribution to a single observation lying inside the simplex. Maximizing the likelihood will lead to a point mass on the observed datapoint (as was also noted by Sadowski and Baldi (2018)); the log-likelihood

[2]Note that the Kth component does not form part of the argument; it is a deterministic function of the $(K-1)$-dimensional random variable $\mathbf{x}$.

will diverge, and so will the parameter estimate (tending to ∞ along the line that preserves the observed proportions). This example may seem degenerate; after all, any dataset that would warrant a probabilistic model had better contain more than a single observation, at which point no individual point can 'pull' the density onto itself. However, in the context of predictive modeling, observations *will* often present unique input-output pairs, particularly if we have continuous predictors, or a large number of categorical ones. Thus, any regression function that is sufficiently flexible, such as a deep network, or that takes a sufficiently large number of inputs, can result in a divergent log-likelihood, frustrating an optimizer's effort to find a sensible estimator. This limitation has constrained Dirichlet-based predictive models to the space of linear functions (Campbell and Mosimann, 1987; Hijazi and Jernigan, 2009).

Second, tails: the Dirichlet density diverges or vanishes at the extrema for all but a set of measure zero values of $\boldsymbol{\alpha}$, so that the log-likelihood is undefined whenever an observation contains zeros (transformation-based alternatives, such as Logratios, suffer the same drawback). However, zeros are ubiquitous in real-world compositional data, a fact that has lead to the development of numerous hacks, such as Palarea-Albaladejo and Martín-Fernández (2008), Scealy and Welsh (2011), Stewart and Field (2011), and Tsagris and Stewart (2018), each of which carries its own tradeoffs.

Third, bias: an elementary result is that the MLE of an exponential family yields an unbiased estimator for its sufficient statistic. However, the sufficient statistic of the Dirichlet distribution is the logarithm of the data, so that by Jensen's inequality, the MLE for the mean parameter, which is often the object of interest, is biased. While the MLE is also asymptotically consistent, this is only the case in the unrealistic setting of a true Dirichlet data-generating process, as we will illustrate empirically in section §4.5.1.

4.3 The Continuous Categorical Distribution

These three limitations motivate the introduction of a novel exponential family, the *continuous categorical* (CC), which is defined on the closed simplex, $\text{cl}(\mathbb{S}^{K-1}) = \{\mathbf{x} \in \mathbb{R}^{K-1} : \mathbf{x} \geq \mathbf{0}, \sum_{i=1}^{K-1} x_i \leq$

1}, by:

$$\mathbf{x} \sim CC(\boldsymbol{\lambda}) \iff p(x_1, \ldots, x_{K-1}; \boldsymbol{\lambda}) \propto \prod_{i=1}^{K} \lambda_i^{x_i}, \tag{4.2}$$

where, again, $x_K = 1 - x_1 - \cdots - x_{K-1}$. Without loss of generality we restrict the parameter values $\{\boldsymbol{\lambda} \in \mathbb{R}_+^K : \sum_i \lambda_i = 1\}$; such a choice makes the model identifiable. The CC density looks much like the Dirichlet, except that we have switched the role of the parameter and the variable. However, this simple exchange results in a well-behaved log-likelihood that can no longer concentrate mass on single interior points (§4.3.1), nor diverge at the extrema (§4.3.2), and that produces unbiased estimators (§4.3.3).

4.3.1 Convexity and Modes

Because the CC log-likelihood is linear in the data (equation 4.2), the density is necessarily convex. It follows that the modes of the CC are at the extrema; example density plots are shown in figure 4.1. In this sense, the CC is a less flexible family than the Dirichlet, as it cannot represent interior modes. In the context of fitting a probability distribution (possibly in a regression setting) to compositional outcomes, however, this choice prevents the CC from concentrating mass on single observations, a fundamental tradeoff with the Dirichlet distribution and other transformation-based competitors such as Aitchison (1982), Aitchison (1994), and Aitchison (1999). The mode of the CC is the basis vector associated with the $\text{argmax}(\lambda_1, \ldots, \lambda_K)$ index of the data, provided the maximizer is unique.

4.3.2 Concentration of Mass

The CC exhibits very different concentration of mass at the extrema relative to the Dirichlet. The former is always finite and strictly positive, whereas the latter either diverges or vanishes for

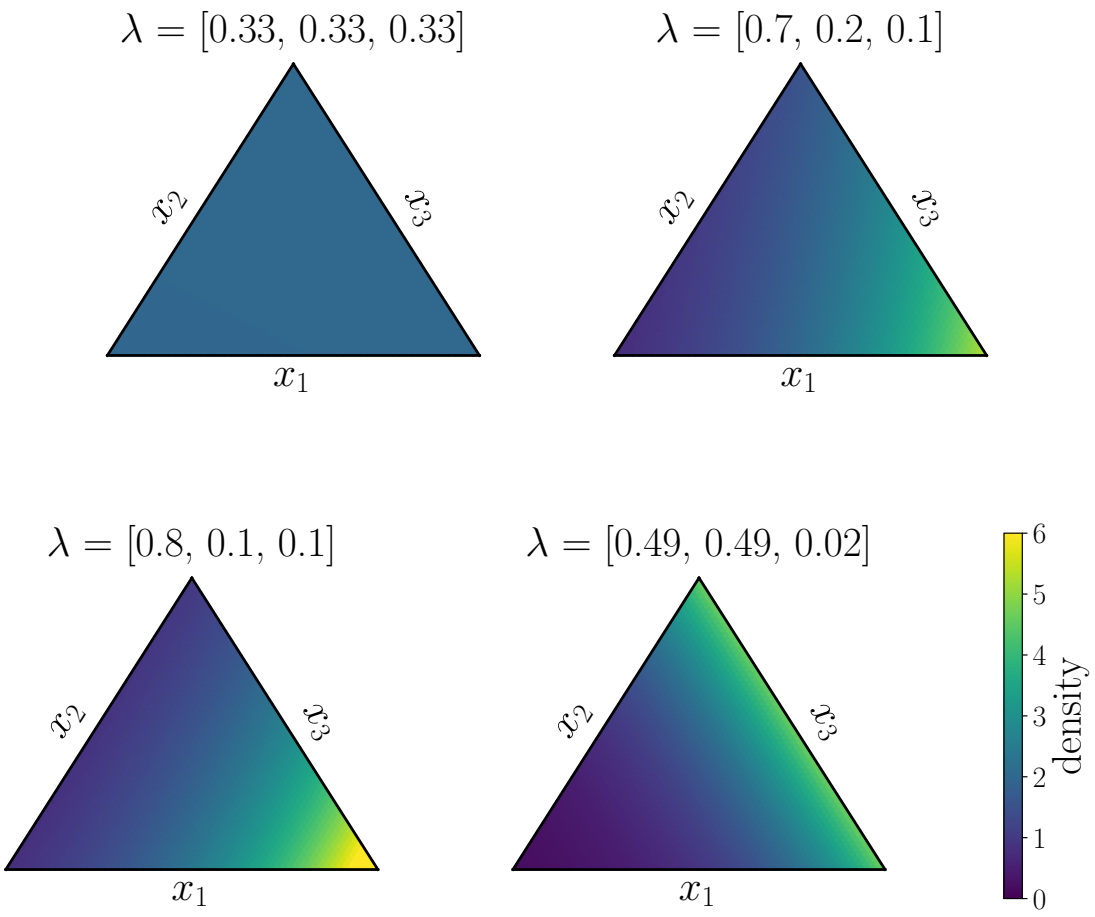


Figure 4.1: Density heatmaps of the 2-dimensional CC (defined on the 3-simplex). We show a near-uniform example, followed by more extremal examples, as well as a bimodal example (with the modes necessarily at the extrema). Note that, while we have defined the CC distribution on the space $\{\mathbf{x} \geq \mathbf{0} : \sum_{i=1}^{K-1} x_i \leq 1\}$, we plot the density on the equivalent set $\{\mathbf{x} \geq \mathbf{0} : \sum_{i=1}^{K} x_i = 1\}$.

almost all parameter values, in other words:

$$\lim_{x_j \to 0} \log \frac{CC(\mathbf{x}|\boldsymbol{\lambda})}{Dirichlet(\mathbf{x}|\boldsymbol{\alpha})} \to \begin{cases} \infty, & \text{if } \alpha_j > 1 \\ -\infty, & \text{if } \alpha_j < 1 \end{cases}. \tag{4.3}$$

The important distinction lies at the limit points; the CC is supported on the *closed* simplex, whereas the Dirichlet and its transformation-based alternatives are defined only on its interior. Thus, the CC log-likelihood can automatically model data with zeros, without requiring specialized techniques such as Palarea-Albaladejo and Martín-Fernández (2008), Scealy and Welsh (2011), Stewart and Field (2011), and Tsagris and Stewart (2018), to name but a few. The sheer amount of published work dedicated to this long-standing issue should convince the reader that this property of the CC distribution provides a substantial advantage in an applied modeling context.

4.3.3 Exponential Family

The CC defines an exponential family of distributions; noting that by definition $x_K = 1 - x_1 - \cdots - x_{K-1}$, we can rewrite equation 4.2:

$$p(\mathbf{x}; \boldsymbol{\lambda}) \propto \exp\left(\sum_{i=1}^{K-1} x_i \log \frac{\lambda_i}{\lambda_K}\right). \tag{4.4}$$

Letting $\eta_i = \log \frac{\lambda_i}{\lambda_K}$, so that $\lambda_i = \frac{\exp(\eta_i)}{\sum_{k=1}^{K} \exp(\eta_k)}$, gives the natural parameter of our exponential family. Under this parameterization, we can ignore the Kth component, $\eta_K = \log \frac{\lambda_K}{\lambda_K} \equiv 0$, and our parameter space becomes the unrestricted $\boldsymbol{\eta} \in \mathbb{R}^{K-1}$, which is more convenient for optimization purposes and will be used throughout our implementation. With a slight abuse of notation,[3] our

[3]We will write $\mathbf{x} \sim CC(\boldsymbol{\lambda})$ and $\mathbf{x} \sim CC(\boldsymbol{\eta})$ interchangeably depending on context, and similarly for the density functions $p(\mathbf{x}; \boldsymbol{\lambda})$ and $p(\mathbf{x}; \boldsymbol{\eta})$.

density simplifies to:

$$p(x_1, \ldots, x_{K-1}; \boldsymbol{\eta}) \propto \exp(\boldsymbol{\eta}^\top \mathbf{x}). \tag{4.5}$$

This last formulation makes it apparent that, under a CC likelihood, the mean of the data is minimal sufficient. By the standard theory of exponential families, this implies that the MLE of the CC distribution will produce an unbiased estimator of the mean parameter. To be precise, if $\hat{\boldsymbol{\lambda}}$ maximizes the CC likelihood, and $\hat{\boldsymbol{\mu}}$ is the corresponding mean parameter obtained from $\hat{\boldsymbol{\lambda}}$, then $\hat{\boldsymbol{\mu}} = \bar{\mathbf{x}}$, where $\bar{\mathbf{x}}$ is the empirical average of the data. Thus, the CC MLE is unbiased for the true mean, irrespective of the data-generating distribution.

This fact stands in contrast to the Dirichlet, in which the sufficient statistic is the mean of the logarithms, or other competitors, in which less can be said about the bias, partly as a result of their added complexities. Not only are these biases undesirable at a philosophical level, but we will also find in section 4.5 that, empirically, they compromise the performance of the Dirichlet relative to the CC.

4.3.4 Normalizing Constant

For our new distribution to be of practical use, we must first derive its normalizing constant, which we denote by $C(\boldsymbol{\eta})$, defined by the equation:

$$\int_{\mathbb{S}^{K-1}} C(\boldsymbol{\eta}) \exp(\boldsymbol{\eta}^\top \mathbf{x}) d\mu(\mathbf{x}) = 1, \tag{4.6}$$

where μ is the Lebesgue measure.

Proposition. The normalizing constant of a $CC(\boldsymbol{\eta})$ random variable is given by:

$$C(\boldsymbol{\eta}) = \left((-1)^{K+1} \sum_{k=1}^{K} \frac{\exp(\eta_k)}{\prod_{i \neq k} (\eta_i - \eta_k)} \right)^{-1}, \tag{4.7}$$

provided $\eta_i \neq \eta_k$ for all $i \neq k$.

Proof. We present a proof by induction in Appendix B.1.

Note it is only on a set of Lebesgue measure zero that $\eta_i = \eta_k$ for some $i \neq k$, and while the CC is still properly defined in these cases, the normalizing constant takes a different form. However, evaluating equation 5.4 when two (or more) parameters are approximately equal can result in numerical instability. In our experiments (§4.5), the limited instances of this issue were dealt with by zeroing out any error-inducing gradients during optimization.

It may come as a surprise that the normalizing constant of the CC distribution (5.4) admits a simple closed form expression in terms of elementary functions; the same cannot be said of almost any multivariable function that one might hope to integrate over the simplex (compare to the Dirichlet density, which integrates to a product of gamma functions). Thus, the CC turns out to be a very natural choice for a distribution on this sample space, which has not been previously proposed in the literature. The appeal of equation 5.4 is not just theoretical; it allows the CC log-likelihood to be optimized straightforwardly using automatic differentiation. In other words, not only is the CC distribution able to address several key limitations of the Dirichlet, but it does so without sacrificing mathematical simplicity (the form of the densities are very similar) or exponential family properties, or adding additional model parameters.

4.3.5 Mean and Variance

By the standard theory of exponential families, the mean and covariance of the CC distribution can be computed by differentiating the normalizing constant. Thus, while cumbersome to write down analytically, we can evaluate these quantities using automatic differentiation. Higher moments, including skewness and kurtosis, can also be derived from the normalizing constant, as well as a number of other distributional results, such as the characteristic function or KL divergence (see Appendix B.2).

4.3.6 Related Distributions

The Dirichlet distribution can be thought of as a generalization of the beta distribution to higher dimensions, which arises by taking the product of independent beta densities and restricting the resulting function to the simplex. In the same way, the CC provides a multivariate extension of the recently proposed continuous Bernoulli (CB) distribution (Loaiza-Ganem and Cunningham, 2019), which is defined on $[0, 1]$ by:

$$x \sim \mathcal{CB}(\lambda) \iff p(x|\lambda) \propto \lambda^x (1-\lambda)^{1-x}. \tag{4.8}$$

First, observe that this density is equivalent to the univariate case of the CC (i.e. $K = 2$). Second, note that the full CC density (4.2) corresponds to the product of independent CB densities, restricted to the simplex, an idea that we will capitalize on when designing sampling algorithms for the CC (§4.4). In this sense, the beta, Dirichlet, CB and CC distributions form an intimately connected tetrad; the CB and the CC switch the role of the parameter and the variable in the beta and Dirichlet densities, respectively, and the Dirichlet and CC extend to the simplex the product of beta and CB densities, respectively. The analogy to the beta and CB families is not just theoretical; the CB arose in the context of Variational Autoencoders (Kingma and Welling, 2014) and showed empirical improvements over the beta for modeling data which lies close to the extrema of the unit interval (Loaiza-Ganem and Cunningham, 2019). Similarly, the CC provides theoretical and empirical improvements over the Dirichlet for modeling extremal data (§4.3.2, 4.5).

4.4 Sampling

Sampling is of fundamental importance for any probability distribution. We developed two novel sampling schemes for the CC, which we highlight here; full derivations and a study of their performance (including reparameterization gradients) can be found in Appendix B.3. We note that, while the Dirichlet distribution can be sampled efficiently via normalized gamma draws or stick-breaking (Connor and Mosimann, 1969), we are not aware of any such equivalents for the CC (see

Appendix B.2.4).

Given the relationship between the CB and the CC densities (§4.3.6), a naive rejection sampler for the CC follows directly by combining independent CB draws (using the closed form inverse cdf), and accepting only those that fall on the simplex. This basic sampler scales poorly in K. We improve upon it via a reordering operation, resulting in vastly better performance (see Appendix B.2.2 and Figure B.1). The central concept of this scheme is to sort the parameter $\boldsymbol{\lambda}$ and reject as soon as samples leave the simplex; this sampler is outlined in algorithm 2.

Algorithm 2 Ordered rejection sampler

Input: target distribution $CC(\boldsymbol{\lambda})$.
Output: sample $\mathbf{x}$ drawn from target.

1: Find the sorting operator π that orders $\boldsymbol{\lambda}$ from largest to smallest, and let $\tilde{\boldsymbol{\lambda}} = \pi(\boldsymbol{\lambda})$.
2: Set the cumulative sum $c \leftarrow 0$ and $i \leftarrow 2$.
3: **while** $c < 1$ and $i \leq K$ **do**
4: Sample $x_i \sim \mathcal{CB}\left(\tilde{\lambda}_i/(\tilde{\lambda}_i + \tilde{\lambda}_1)\right)$.
5: Set $c \leftarrow c + x_i$ and
6: Set $i \leftarrow i + 1$.
7: **end while**
8: If $c > 1$, go back to step 2.
9: Set $x_1 = 1 - \sum_{i=2}^{K} x_i$.
10: Return $\mathbf{x} = \pi^{-1}(x_1, \ldots, x_K)$.

Algorithm 2 performs particularly well in the case that $\boldsymbol{\lambda}$ is unbalanced, with a small number of components concentrating most of the mass. However, its efficiency degrades under balanced configurations of $\boldsymbol{\lambda}$; this shortcoming motivates the need for our second sampler (algorithm 7), which performs particularly well in the balanced setting. Conceptually, this second sampler will exploit a permutation-induced partition of the unit cube into simplices, combined with a relaxation of the CC which is invariant under permutations, and can be mapped back to the original CC distribution.

At a technical level, first note that if σ is a permutation of $\{1, \ldots, K-1\}$ and $\mathcal{S}_\sigma = \{\mathbf{x} \in \mathbb{R}^{K-1} : 0 \leq x_{\sigma(1)} \leq x_{\sigma(2)} \leq \cdots \leq x_{\sigma(K-1)} \leq 1\}$, then we can (almost, in the measure-theoretic sense) partition $[0,1]^{K-1} = \bigcup \mathcal{S}_\sigma$ where the union is over all permutations. Second, we can generalize the CC to an arbitrary sample space by writing $p_\Omega(\mathbf{x}|\boldsymbol{\eta}) \propto \exp(\boldsymbol{\eta}^\top \mathbf{x}) \mathbb{1}(\mathbf{x} \in \Omega)$. By the change of

variable formula, we note that this family is invariant to invertible linear maps in the sense that, if $\mathbf{x} \sim p_{\mathcal{A}}(\mathbf{x}|\boldsymbol{\eta})$ and $\mathbf{y} = Q\mathbf{x}$, where Q is an invertible matrix, then $\mathbf{y} \sim p_{Q(\mathcal{A})}(\mathbf{y}|\tilde{\boldsymbol{\eta}})$, where $\tilde{\boldsymbol{\eta}} = Q^{-\top}\boldsymbol{\eta}$. Hence, if B is a lower-triangular matrix of ones and id denotes the identity permutation, then $\mathcal{S}_{id} = B(\mathrm{cl}(\mathbb{S}^{K-1}))$, and it follows that we can sample from $\mathbf{x} \sim CC(\boldsymbol{\eta})$ by sampling $\mathbf{y} \sim p_{\mathcal{S}_{id}}(\mathbf{y}|\tilde{\boldsymbol{\eta}})$ and transforming with $\mathbf{x} = B^{-1}\mathbf{y}$. Conveniently, $\mathbf{y} \sim p_{\mathcal{S}_{id}}(\mathbf{y}|\tilde{\boldsymbol{\eta}})$ can be sampled by first drawing a $[0,1]^{K-1}$-valued vector of independent CB variates, namely $\mathbf{y}' \sim p_{[0,1]^{K-1}}(\mathbf{y}'|\tilde{\boldsymbol{\eta}})$, then transforming into $\mathcal{S}_{id}$ by applying $\mathbf{y} = P\mathbf{y}'$, where P is the permutation matrix that orders the elements of $\mathbf{y}'$, and finally accepting $\mathbf{y}$ with probability:

$$\alpha(\mathbf{y}, \tilde{\boldsymbol{\eta}}, P) = \frac{p_{\mathcal{S}_{id}}(\mathbf{y}|\tilde{\boldsymbol{\eta}})}{\kappa(\tilde{\boldsymbol{\eta}}, P) p_{\mathcal{S}_{id}}(\mathbf{y}|P^{-\top}\tilde{\boldsymbol{\eta}})}, \tag{4.9}$$

where $\kappa(\tilde{\boldsymbol{\eta}}, P)$ is the rejection sampling constant:

$$\kappa(\tilde{\boldsymbol{\eta}}, P) = \max_{\mathbf{y} \in \mathcal{S}_{id}} \frac{p_{\mathcal{S}_{id}}(\mathbf{y}|\tilde{\boldsymbol{\eta}})}{p_{\mathcal{S}_{id}}(\mathbf{y}|P^{-\top}\tilde{\boldsymbol{\eta}})}. \tag{4.10}$$

This sampling scheme is shown altogether in algorithm 7. It performs particularly well in the case that $\boldsymbol{\lambda}$ is balanced, since the distributions induced on $\mathcal{S}_{id}$ by $\tilde{\boldsymbol{\eta}}$ and the permutations thereof, are similar, resulting in high acceptance probabilities. Taken together with algorithm 2, our permutation sampler provides an efficient, theoretically understood, and reparameterizable sampling scheme for the CC.

Algorithm 3 Permutation sampler

Input: target distribution $CC(\boldsymbol{\eta})$.
Output: sample $\mathbf{x}$ drawn from target.

1: Sample $\mathbf{y}' \sim p_{[0,1]^{K-1}}(\cdot|\tilde{\boldsymbol{\eta}})$, where $\tilde{\boldsymbol{\eta}} = B^{-\top}\boldsymbol{\eta}$.
2: Let $\mathbf{y} = P\mathbf{y}'$, where P is the permutation matrix that sorts $\mathbf{y}$ in increasing order.
3: With probability $\alpha(\mathbf{y}, \tilde{\boldsymbol{\eta}}, P)$, accept $\mathbf{y}$ and return $\mathbf{x} = B^{-1}\mathbf{y}$. Otherwise, go back to step 1.

4.5 Experiments

4.5.1 Simulation Study

We begin our experiments with a simulation study that illustrates the biases incurred from fitting Dirichlet distributions to compositional data. Our procedure is as follows:

1. Fix a ground-truth distribution $\mathbf{x} \sim p(\mathbf{x})$ on the simplex. This will either be a Dirichlet where the (known) parameter value $\boldsymbol{\alpha}$ is drawn from independent Exp(1) variates, or a CC where the (known) parameter value $\boldsymbol{\lambda}$ is sampled uniformly on the simplex.
2. Fix a sample size n. This will range from $n = 2$ to $n = 20$.
3. In each of one million trials, draw a sample of n i.i.d. observations $\mathbf{x}_i \sim p(\mathbf{x})$.
4. Use the samples to compute one million MLEs for the mean parameter, under both a Dirichlet and a CC likelihood.
5. Average the one million MLEs and subtract the true mean $\mathbb{E}_{p(\mathbf{x})}[\mathbf{x}]$ to obtain estimates of the 'empirical bias'.
6. Repeat the steps for different values of $\boldsymbol{\alpha}$, $\boldsymbol{\lambda}$, and n, as prescribed.

The results of this experiment are shown in figure 4.2. As we already knew from the theory of exponential families, only the CC estimator is unbiased. The Dirichlet estimator is, at best, asymptotically unbiased, and only in the unrealistic setting of a true Dirichlet generative process (pink line). It is worth emphasizing that, even in this most-favorable case for the Dirichlet, the CC outperforms (pink line versus blue line). The error bars show ±1 standard deviation over different draws from the prior distribution of step 1. The large error bars on the Dirichlet, especially under non-Dirichlet data (orange line) indicate that its bias is highly sensitive to the true mean of the distribution, reaching up to several percentage points (relative to the unit-sum total), whereas the CC is unbiased across the board. Lastly, note that while a sample size of 20 may seem small in the context of machine learning, this is in fact reasonable in terms of the ratio of observations to parameters, as larger samples typically call for more flexible models.

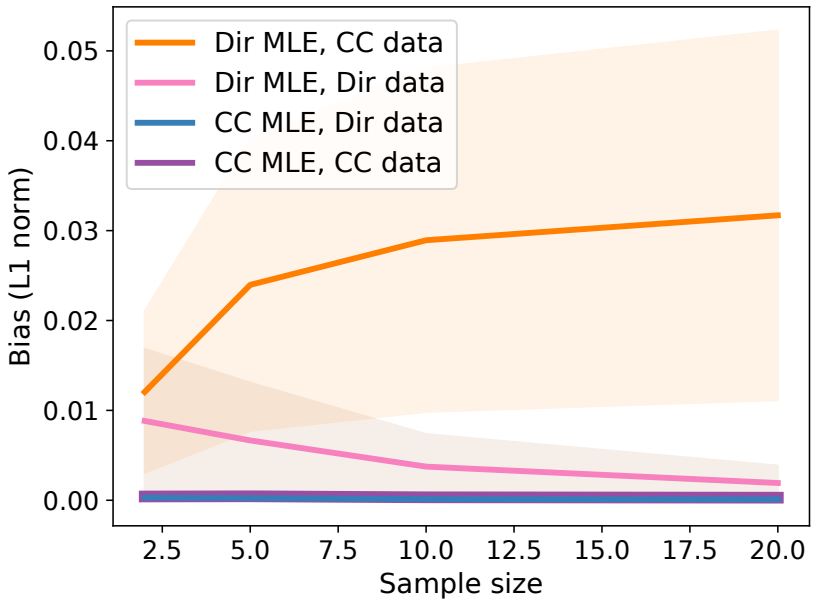


Figure 4.2: Empirical bias of the Dirichlet and CC MLEs as a function of sample size, for $K = 3$ (other values of K behaved similarly). The error bars show ± 1 standard deviation over different draws of the parameters of the ground-truth model from their respective priors. Regardless of whether the synthetic data is generated from a Dirichlet or a CC, only the CC estimator is unbiased.

4.5.2 UK Election Data

We next consider a real-world example from the 2019 UK general election (Uberoi, Baker, and Cracknell, 2019). The UK electorate is divided into 650 constituencies, each of which elects 1 member of parliament (MP) in a winner-takes-all vote. Typically, each of the major parties will be represented in each constituency, and will win a share of that constituency's electorate. Thus, our data consists of 650 observations[4], each of which is a vector of proportions over the four major parties (plus a fifth proportion for a 'remainder' category, which groups together smaller parties and independent candidates). We regress this outcome on four predictors: the region of the constituency (a categorical variable with 4 levels), an urban/rural indicator, the size of the constituency's electorate, and the voter turnout percentage from the previous general election. While there is a host of demographic data that could be used to construct increasingly more informative

[4]We split the data into an 80/20 training and test set at random.

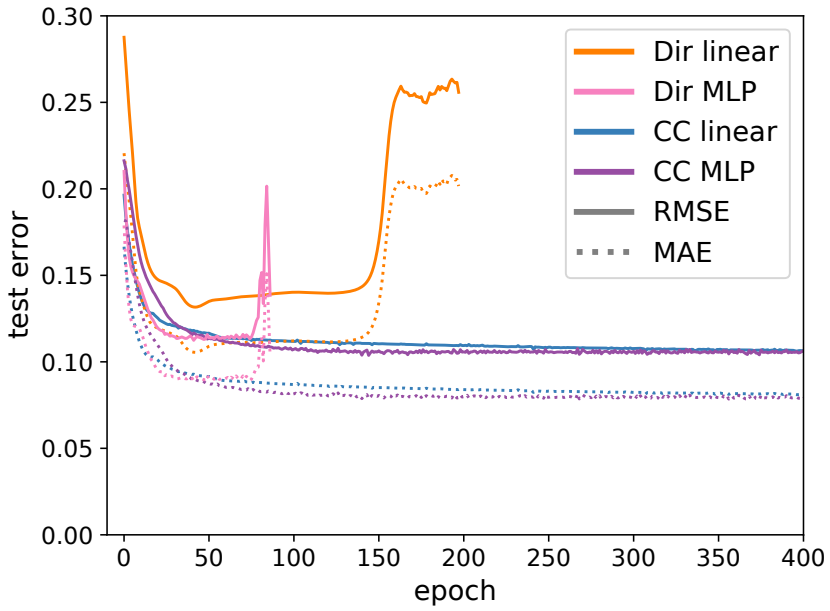


Figure 4.3: Test error for the linear and MLP models with a Dirichlet and CC log-likelihood. The CC objective is better-behaved, training more smoothly and converging to the MLE, unlike the Dirichlet log-likelihood which diverges, causing overfitting.

predictors here, we stick to the reduced number of variables that were available in our original dataset, as the goal of our analysis is *not* to build a strong predictive model, but rather to illustrate the advantages and disadvantages of using the novel CC distribution as opposed to the Dirichlet. For the benefit of the latter, since our data contains zeros (not all major parties are represented in all constituencies), we add an insignificant 0.1% share to all observations and re-normalize prior to modeling (without this data distortion, the Dirichlet would fail even more grievously).

We fit regression networks to this data with two different loss functions, one where we assume the response follows a Dirichlet likelihood (Campbell and Mosimann, 1987; Hijazi, 2003; Hijazi and Jernigan, 2009), the other with our own CC likelihood instead. We first fit the simplest linear version: for the Dirichlet, we map the inputs into the space of $\boldsymbol{\alpha}$ by applying a single single linear transformation followed by an $\exp(\cdot)$ activation function; for the CC, a single linear transformation is sufficient to map the inputs to the unconstrained space of $\boldsymbol{\eta}$ (in statistical terminology, this corresponds to a generalized linear model with canonical link). Secondly, we extend our linear model

Table 4.1: Test errors and runtime for our regression models of the UK election data. Both in the linear and MLP case, the CC model beats the Dirichlet counterpart.

	MAE	RMSE	MS/EPOCH
DIR LINEAR	0.105	0.132	4
CC LINEAR	**0.075**	**0.101**	8
DIR MLP	0.087	0.112	20
CC MLP	**0.072**	**0.097**	35

to a more flexible neural network by adding a hidden layer with 20 units and ReLU activations. We train both models using Adam (Kingma and Ba, 2015).

The CC models achieve better test error, with gains ranging between 10% and 30% in the L_1 and L_2 sense, as shown in table 4.1 and figure 4.3.[5] Moreover, the CC reliably converges to a performant model (which in the linear case corresponds to the MLE), whereas the Dirichlet likelihood diverges along a highly variable path in the parameter space that correspond to suboptimal models. We verify also that this behavior happens irrespective of the optimizer used to train the model (figure 4.4), and is consistent across random initializations of the model parameters (figure 4.5). Note that the Dirichlet MLP diverges faster than its linear counterpart, likely because the increased flexibility afforded by the MLP architecture allows it to place a spike on a training point more quickly.

Naturally, one might ask whether the unstable behavior of the Dirichlet can be fixed through regularization, as a suitable penalty term might be able to counterbalance the detrimental flexibility of the distribution. We test this hypothesis empirically by adding an L_2 penalty (applied to the weights and biases of the regression network) to the objective, with a varying coefficient to control the regularization strength. The results are shown in figure 4.6; while strong regularization does stabilize the Dirichlet, it is still unable to outperform the (unregularized) CC and it is slower to converge.

In terms of runtime, we find that the computational cost of the CC and Dirichlet gradients are

[5]The L_1 and L_2 metrics are used for illustration purposes. One could, of course, optimize these metrics directly, however this would either jeopardize the probabilistic interpretation and statistical rigor of the model, or require the addition of an intractable normalizing constant to the log-likelihood.

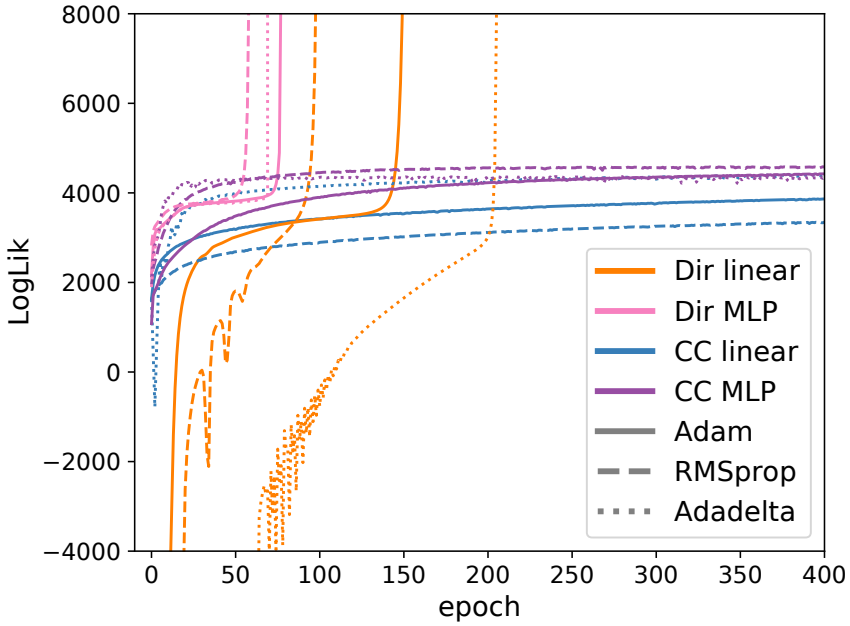


Figure 4.4: Log-likelihood during training with different optimizers for the Dirichlet and the CC models. Consistently across optimizers, the CC objective converges to the MLE and the Dirichlet diverges.

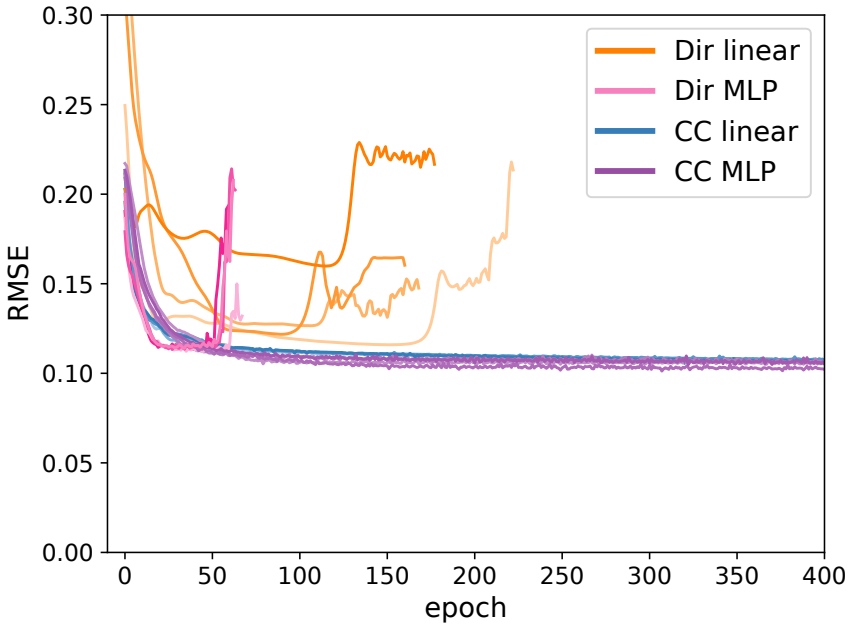


Figure 4.5: Test error for different random initializations of the model parameters. The CC model converges, whereas the Dirichlet diverges along a highly variable path in the parameter space, and the models it finds in this path are suboptimal.

of the same order of magnitude (table 4.1). While the CC models are slower per gradient step, they are able to find better regression functions in fewer steps. What's more, the comparison is somewhat unfair to the CC, as the Dirichlet gradients exploit specialized numerical techniques for evaluating the digamma function that have been years in the making, whereas the same is not yet true for our novel distribution.

4.5.3 Knowledge Distillation

Our last experiment considers a typical model compression task (Buciluǎ, Caruana, and Niculescu-Mizil, 2006; Ba and Caruana, 2014; Hinton, Vinyals, and Dean, 2015), where we have access to an accurate 'teacher' model that is expensive to evaluate, which we use in order to train a cheaper 'student' model, typically a neural network with few layers. By training the student network to predict the fitted values of the teacher model, it can achieve better generalization than when trained on the original labels, as was shown by Buciluǎ, Caruana, and Niculescu-Mizil (2006), Ba and Caruana (2014), and Hinton, Vinyals, and Dean (2015). These fitted values provide 'soft targets' that are typically located close to the extrema of the simplex, though we can also bring them towards the centroid while conserving their relative order by varying the temperature, T, of the final softmax (Hinton, Vinyals, and Dean, 2015).

We build on the MNIST experiment of Hinton, Vinyals, and Dean (2015); we first train a teacher neural net with two hidden layers of 1200 units and ReLU activations, regularized using batch normalization, and we use its fitted values to train smaller student networks with a single hidden layer of 30 units. We fit the student networks to the soft targets using the categorical cross-entropy (XE) loss, as well as a Dirichlet (as per Sadowski and Baldi (2018)) and a CC log-likelihood. The latter is equivalent to adding the appropriate normalization constant to the XE, thus giving a correct probabilistic interpretation to the procedure. Note that the XE and the CC objective result in the same estimator at optimality: the empirical mean of the data (we know this is true theoretically from section 4.3.3). However, the two objectives define different optimization landscapes, so the question of which one to use becomes an empirical one.

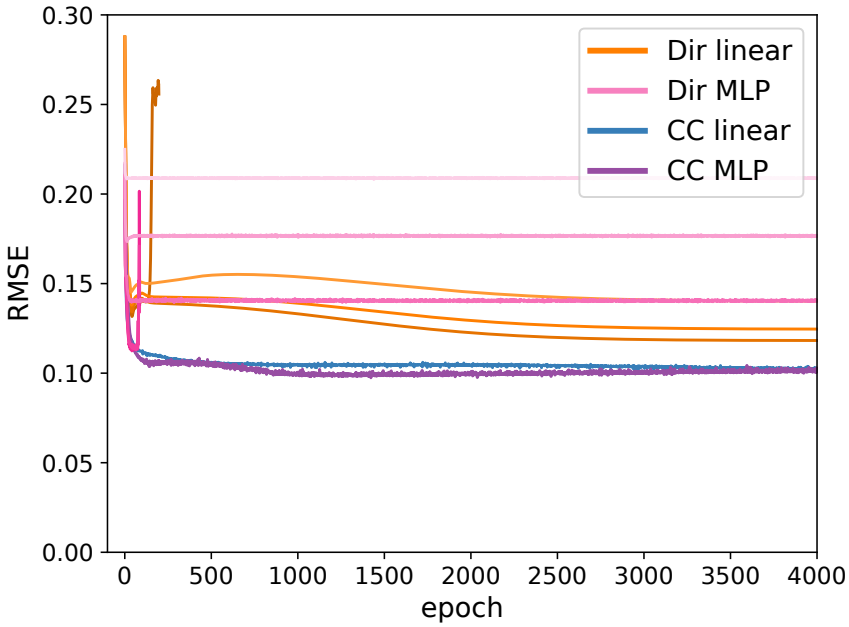


Figure 4.6: Test error for different regularization strengths for the Dirichlet models. Lighter shades of pink and orange indicate increasing regularization strengths. While sufficiently strong regularization does stabilize the Dirichlet, preventing it from diverging, it still does not outperform the (unregularized) CC.

In this case, we found that using the CC objective leads to a lower misclassification rate on the test set than the XE (table 4.2), as well as a lower RMSE (measured against the soft targets outputted by the trained teacher model when evaluated on the test set), and did so consistently across temperatures (figure 4.7). Both the CC and XE objectives worked substantially better than the Dirichlet likelihood, and better than training the same architecture on the original hard labels. Note also that the Dirichlet performs very poorly and is unstable in the unadjusted temperature setting ($T = 1$), as the soft targets are very extremal and lead to numerical overflow, but performs much better at high temperatures.

4.5.4 Latent Variable Models

While we have demonstrated empirical improvements from using the CC to model compositional outcomes, we have found no such gains when using the CC to model latent variables, for example, in mixed-membership models such as LDA (Blei, Ng, and Jordan, 2003; Pritchard,

Table 4.2: Test errors for the student models trained under the Dirichlet, XE, and CC objectives, as well as using the hard labels instead of the teacher model. We show the best values obtained over 5 random initializations and the 3 temperature settings of figure 4.7.

OBJECTIVE	ACCURACY	RMSE	S/EPOCH
DIRICHLET	90.6%	0.041	0.9
SOFT XE	94.9%	0.029	0.8
CC	**95.6%**	**0.024**	2.2
HARD XE	93.2%	–	0.8

Stephens, and Donnelly, 2000). Training a 'Latent-CC-Allocation' topic model on a corpus of ~2,000 NeurIPS papers gave similar learned topics and held-out perplexities as vanilla LDA (562 vs 557 per word, respectively). However, while the Dirichlet is the conjugate prior for the multinomial, a CC prior with a multinomial likelihood leads to an intractable posterior, complicating optimization (which, in this case, is enabled by means of our sampling and reparameterization algorithms from appendix B.3). We note also that the CC does not share the sparsity-inducing properties of the Dirichlet, nor can it approximate the categorical distribution arbitrarily well, unlike other continuous relaxations that have been proposed in the literature (Jang, Gu, and Poole, 2016; Maddison, Mnih, and Teh, 2016; Potapczynski, Loaiza-Ganem, and Cunningham, 2020). Nevertheless, given the widespread use of simplex-valued latent variables in probabilistic generative models, the use of the CC distribution in this context remains an open question.

4.6 Conclusion

Our results demonstrate the theoretical and empirical benefits of the CC, which should hold across a range of applied modeling contexts. To conclude, we summarize the main contributions of our work:

- We have introduced the CC distribution, a novel exponential family defined on the simplex, that resolves a number of long-standing limitations of previous models of compositional data.

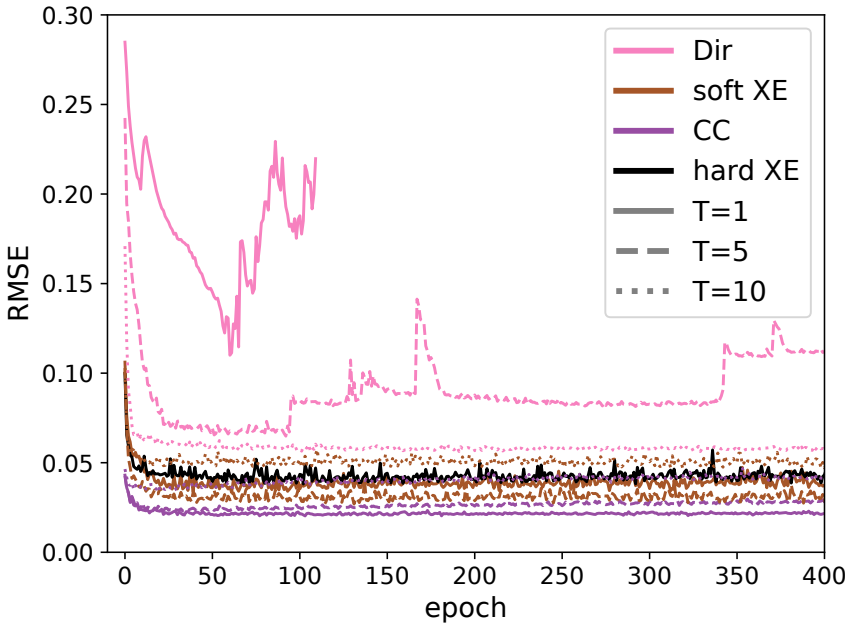


Figure 4.7: Test error at different temperatures for the Dirichlet, XE and CC objectives. For each student model, this error measures the L_2 difference between its fitted values and those of the teacher model, on the test set.

- We have fully characterized our distribution and discussed its theoretical properties. Of particular importance is the sufficient statistic, which guarantees unbiased estimators, and the favorable behavior of the log-likelihood, which leads to ease of optimization and robustness to extreme values.
- Empirically, the CC defines probabilistic models that outperform their Dirichlet counterparts and other competitors, and allows for a rich class of regression functions, including neural networks.
- We have also designed and implemented novel and efficient rejection sampling algorithms for the CC.

Taken together, these findings indicate the CC is a valuable density for probabilistic machine learning with simplex-valued data.

Chapter 5: Uses and Abuses of the Cross Entropy Loss

Escucha canijo. Escucha primo.
Escucha lo que te digo.
Aquí antes que amapolas, picha,
somos trigo.

Migue Benítez

5.1 Introduction

Modern deep learning is primarily an experimental science, in which empirical advances occasionally come at the expense of probabilistic rigor. Here we focus on one such example; namely the use of the categorical cross-entropy loss to model data that is not strictly categorical, but rather takes values on the simplex. This practice is standard in neural network architectures with label smoothing and actor-mimic reinforcement learning, amongst others. Drawing on our continuous categorical distribution from Chapter 4, we propose probabilistically-inspired alternatives to these models, providing an approach that is more principled and theoretically appealing. Through careful experimentation, including an ablation study, we identify the potential for outperformance in these models, thereby highlighting the importance of a proper probabilistic treatment, as well as illustrating some of the failure modes thereof.[1]

The cross-entropy loss is one of the most commonly used loss functions for training deep neural network models, most notably in (multi-class) classification problems. When applied to categorical data, this loss function corresponds to a probabilistic log-likelihood, therefore resulting in favorable estimation properties. On the other hand, several prominent methods in modern

[1] Our code is available at `https://github.com/cunningham-lab/cb_and_cc`.

machine learning are concerned with fitting data that is not quite categorical, but simplex-valued; key examples being the "soft targets" in label smoothing (LS) (Szegedy et al., 2016), and "expert policies" in actor-mimic reinforcement learning (AMN) (Parisotto, Ba, and Salakhutdinov, 2015), amongst others (Hinton, Vinyals, and Dean, 2015; Tzeng et al., 2015). In these methods, the deep learning community has defaulted to borrowing the same cross-entropy loss from the categorical case, despite the fact that it no longer defines a bona fide probability model. As well as highlighting this practice and putting it into question, our work proposes adjusting the LS and AMN objective functions by replacing the cross-entropy loss with the log-likelihood of the continuous categorical (CC) distribution. Doing so amounts to incorporating a normalizing constant to our model, or in other words, adding the factor that scales the cross-entropy loss to a valid probability density function over the simplex. As of yet, such an approach has only been considered in the context of knowledge distillation (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020), although the one-dimensional special case (corresponding to the binary cross-entropy with $[0, 1]$-valued data) has been studied more extensively (Loaiza-Ganem and Cunningham, 2019). Our inspiration draws from both of these works, although our focus is primarily on LS and AMN architectures instead.

This chapter, which was published as (Gordon-Rodriguez et al., 2020), is organized as follows (note that our two main Sections, 5.3 and 5.4, are based on the same idea, but are broadly independent of one another and can be read separately):

- In Section 5.2 we detail the relevant background on the continuous categorical distribution, highlighting its close connection to the cross-entropy loss.

- Section 5.3 focuses on label smoothing. We propose a novel CC-LS model and perform an ablation study to isolate its potential as a regularizer for classification networks, as well as a qualitative assessment of its learned representations.

- Section 5.4 focuses on actor-mimic reinforcement learning. We recast the AMN model as the solution to a regression problem of simplex-valued data, we propose a novel CC-AMN model, and we provide an experimental evaluation thereof.

- Section 5.5 concludes, combining insights from CC-LS and CC-AMN, and discussing potential directions for future research.

5.2 From the Cross-Entropy to the Continuous Categorical

We briefly review the categorical cross-entropy loss, in order to highlight the close connection with the continuous categorical distribution of Chapter 4. Note that, in line with the applications presented in this Chapter, we shall use y to denote a categorical or simplex-valued random variable, departing slightly from the notation of Chapter 4.

Categorical data refers to observations y that take values in a discrete sample space Ω formed by K distinct elements, which are typically expressed using the K one-hot vectors that form the standard basis of $\mathbb{R}^K$, namely $\Omega = \{e_1, \dots e_K\}$, where $(e_k)_j = \mathbb{1}(k = j)$. In this notation, the cross-entropy loss is equivalent to the negative log-likelihood of $y \in \Omega$ under a categorical distribution with parameter π:

$$l(\pi; y) = -\sum_{k=1}^{K} y_k \log \pi_k \iff p(y; \pi) = \prod_{k=1}^{K} \pi_k^{y_k}. \tag{5.1}$$

In other words, the cross-entropy loss defines a coherent probabilistic model for discrete data over K classes. This elementary fact should not be overlooked; it provides the benefits of the theory of maximum likelihood estimation, including frequentist consistency and asymptotic efficiency, as well as enabling efficient Bayesian inference by specifying a conjugate prior.

So far so good. However, what happens when the observation is not quite categorical, but instead takes values on the simplex, $\Delta^K = \{y \in \mathbb{R}_+^K : \sum_{k=1}^K y_k = 1\}$? Such data plays a key role in label smoothing (Szegedy et al., 2016), actor-mimic reinforcement learning (Parisotto, Ba, and Salakhutdinov, 2015), knowledge distillation (Hinton, Vinyals, and Dean, 2015), and domain adaptation (Tzeng et al., 2015). In all of these methods, neural networks are trained to target a simplex-valued outcome, $y \in \Delta^K$, using the cross-entropy loss $l(\lambda; y) = -\sum_{k=1}^K y_k \log \lambda_k$, where λ represents the output of a neural network. Crucially though, the change in sample space from Ω

to Δ^K breaks the equivalence in (5.1) because the right-hand expression no longer defines a proper probability distribution; its integral over Δ^K does not normalize to 1.

Given the attractive properties of maximum likelihood estimation, there are still good reasons why a legitimate probability model is desirable (see (Loaiza-Ganem and Cunningham, 2019) for a more detailed discussion). The classical statistics literature offers some possibilities, notably the use of *logratios* (Aitchison, 1982; Aitchison, 1994; Aitchison, 1999; Egozcue et al., 2003), or *Dirichlet regression* (Campbell and Mosimann, 1987; Hijazi and Jernigan, 2009). However, we argue that the most natural probabilistic solution is to apply the continuous categorical distribution, since this corresponds to normalizing the cross-entropy loss directly so that it becomes a genuine log-likelihood model, namely:

$$l(\lambda; y) = -\log C(\lambda) - \sum_{k=1}^{K} y_k \log \lambda_k \iff p(y; \lambda) = C(\lambda) \cdot \prod_{k=1}^{K} \lambda_k^{y_k}, \tag{5.2}$$

where $C(\lambda)$ is the normalizing constant:

$$C(\lambda) = \left(\int_{\Delta^K} \prod_{k=1}^{K} \lambda_k^{y_k} dy_k \right)^{-1}. \tag{5.3}$$

As discussed in the previous Chapter, this distribution possesses a number of attractive theoretical and empirical properties (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020); we restate the closed form expression of its normalizing constant:

$$C(\lambda) = \left((-1)^{K+1} \sum_{k=1}^{K} \frac{\lambda_k}{\prod_{i \neq k} \log \frac{\lambda_i}{\lambda_k}} \right)^{-1}. \tag{5.4}$$

This expression is of particular importance to the present Chapter, as it enables the use of automatic differentiation for optimizing models with the continuous categorical log-likelihood (5.2).

5.3 Continuous Categorical Label Smoothing

Label smoothing (Szegedy et al., 2016) has enjoyed rapid growth and widespread use as a means to reduce overfitting and improve the out-of-sample accuracy of neural network classifiers across a range of tasks including computer vision (Zoph et al., 2018; Real et al., 2019), speech recognition (Chorowski and Jaitly, 2016), and machine translation (Vaswani et al., 2017). The mechanism is simple: given a neural network classifier $f_\theta : x \to y$, we replace our one-hot labels $y \in \Omega$ with "soft" targets:

$$y^{\text{LS}} = (1 - \varepsilon)y + \varepsilon u, \tag{5.5}$$

where $u = (1/K, \ldots, 1/K)^\top$ is a uniform vector and $\varepsilon > 0$ is a constant. The network weights θ are then trained to minimize the cross-entropy loss between the network output $f_\theta(x)$ and the modified data y^{LS}.

Equation 5.5 maps $y \in \Omega$ to $y^{\text{LS}} \in \Delta^K$, so that our targets are no longer categorical, but simplex-valued. Thus, even though they are not continuously distributed, it is natural to consider label-smoothed classification through the lens of compositional regression. Our proposal is therefore to use a continuous categorical log-likelihood in lieu of the cross-entropy loss, and we refer to this model as CC-LS. Namely, we are interested in comparing the usual label smoothing loss:

$$\min_\theta \mathcal{L}^{\text{LS}}(\theta) = -\sum_{(x,y)} \sum_k y_k^{\text{LS}} \cdot \log[f_\theta(x)]_k, \tag{5.6}$$

against its continuous categorical counterpart:

$$\min_\theta \mathcal{L}^{\text{CC-LS}}(\theta) = -\sum_{(x,y)} \left\{ \log C(f_\theta(x)) + \sum_k y_k^{\text{LS}} \cdot \log[f_\theta(x)]_k \right\}. \tag{5.7}$$

We remark that, strictly speaking, in order to make our targets continuous over the simplex, we would also have to add continuous noise to the labels, for example by drawing u uniformly

at random on the simplex. Such an approach produced little difference over using the fixed value $u = (1/K, \ldots, 1/K)^\top$, neither in LS nor CC-LS, and we will omit the results for clarity. However, given the wealth of existing methods that achieve improved generalization error by adding noise at different stages in the training procedure (Bishop, 1995; Srivastava et al., 2014; Shorten and Khoshgoftaar, 2019), the idea of smoothing the labels with random noise may still hold potential, and we leave its further analysis for future work.

5.3.1 Experiments

Following the experimental setup of Muller et al (Müller, Kornblith, and Hinton, 2019), we train a CNN classifier on CIFAR-10, with and without label smoothing as well as our novel CC-LS model (see Appendix C.1.1 for the full details of our architecture). This is an example in which label smoothing provided no significant gain over the un-smoothed baseline, likely because the CNN is already regularized using dropout (Srivastava et al., 2014), weight decay (Krogh and Hertz, 1992), and batch normalization (Ioffe and Szegedy, 2015), which altogether are sufficient to provide a good model of the data, given the level of complexity of CIFAR-10. Likewise, we find that the CC-LS model also performs no better than the baseline in this setting (top row of Table 5.1).

Driven by these observations, we perform an ablation study over the different regularizers used in our network, and the results paint a more interesting picture (Table 5.1). Notably, we find that for the unregularized CNN (bottom row), CC-LS significantly outperforms both LS and the baseline. In the case where our network is partially regularized with dropout only, the baseline becomes equally good as LS, but the gap with CC-LS remains wide (penultimate row), and the gain from CC-LS persists after adding weight decay. On the other hand, batch normalization (top half) was sufficient to capture all the gain in test accuracy, with neither LS nor CC-LS outperforming the baseline in these cases. Under weight decay without batch normalization (rows 5 and 6), training became less stable (as evidenced by the large standard deviations), but CC-LS was able to reduce the variability in model accuracy. Overall, Table 5.1 indicates that the CC-LS loss function pro-

Table 5.1: Ablation study for label smoothing on CIFAR-10. We show out-of-sample accuracy for our baseline classifier (w/o LS), as well as vanilla LS and CC-LS, both with $\varepsilon = 0.1$. Errors indicate the standard deviation over 10 random initializations of the network. We consider the effect of LS and CC-LS over the baseline under each combination of dropout, weight decay and batch normalization, and find that CC-LS provides significant outperformance in the absence of BatchNorm.

Dropout	Weight decay	BatchNorm	w/o LS	with LS	CC-LS
Yes	Yes	Yes	**89.5** (±0.1)	89.1 (±0.2)	89.0 (±0.2)
No	Yes	Yes	**89.6** (±0.1)	89.2 (±0.1)	89.2 (±0.2)
Yes	No	Yes	**89.4** (±0.2)	89.3 (±0.2)	89.0 (±0.2)
No	No	Yes	**89.5** (±0.2)	89.4 (±0.1)	89.1 (±0.2)
Yes	Yes	No	88.6 (±1.2)	88.6 (±1.0)	88.7 (±0.6)
No	Yes	No	88.8 (±1.2)	88.7 (±1.0)	88.6 (±0.6)
Yes	No	No	87.0 (±0.2)	87.0 (±0.1)	**87.6** (±0.2)
No	No	No	86.8 (±0.1)	87.0 (±0.2)	**87.6** (±0.2)

Table 5.2: Ratio of within-cluster sum of squares over between-cluster sum of squares, for the learned representations of Figure 5.1. Each cell shows the mean ratio over 10 random initializations, with standard errors.

Samples	w/o LS	with LS	CC-LS
Training	18% (±1)	9% (±1)	12% (±1)
Test	25% (±1)	20% (±1)	23% (±1)

vides a different (and sometimes, significantly better) regularization effect than that of vanilla LS, suggesting its potential for novel applications, particularly in the settings where batch normalization may be undesirable (Galloway et al., 2019). We note further that numerous existing works have been devoted to analyzing the interplay between dropout, weight decay, and batch normalization (Van Laarhoven, 2017; Garbin, Zhu, and Marques, 2020; Chen et al., 2019; Li et al., 2019; Hernández-García and König, 2018); our focus is specifically on their relation to label smoothing and CC-LS.

We end this section with a qualitative analysis of the learned representations from our trained classifiers. Again following (Müller, Kornblith, and Hinton, 2019), we define the "template" vector of the kth class, w_k, as the weight vector from the last CNN layer that is associated to the kth class,

so that in other words:

$$[f_\theta(x)]_k = \frac{e^{w_k^\top z}}{\sum_{k'} e^{w_{k'}^\top z}}, \tag{5.8}$$

where z is a vector containing the activations from the penultimate layer. We then fix three classes, and construct an orthonormal basis (consisting of two vectors) for the plane containing their three template vectors. For each of the classes, we pick a random sample of input data belonging to that class and project their penultimate layer activations onto this plane. The results are shown in Figure 5.1 for the (arbitrarily chosen) classes "airplane", "automobile", and "bird". As was noted by (Müller, Kornblith, and Hinton, 2019), while label smoothing can help the classifier achieve better accuracy on the test set, it comes at the cost of a less informative learned representation, as can be seen from the more concentrated centroids in the second column relative to the first. On the other hand, CC-LS achieves a somewhat richer representation than vanilla LS, as can be observed from the greater within-cluster variances in the third column, which we quantify in Table 5.2. This suggests that the CC may offer additional potential for combining LS with teacher models in the context of knowledge distillation, a setting in which the concentrated clusters enforced by LS proved detrimental to the training of a student model (Müller, Kornblith, and Hinton, 2019).

5.4 Probabilistic Actor-Mimic Reinforcement Learning

In this section we summarize the Actor-Mimic Reinforcement Learning framework (Parisotto, Ba, and Salakhutdinov, 2015) and recast it as a compositional regression problem, highlighting the potential for a probabilistic model with the continuous categorical distribution.

Actor-Mimic Networks (AMN) provide a method for multitask and transfer reinforcement learning. The goal of the AMN is to train a single agent to perform on several different "source games", $\{G_1, \ldots, G_L\}$, each of which corresponds to a Markov Decision Process defined on a common state space and action set (shared across source tasks), but driven by a different set of transition probabilities and reward functions (specific to each task). Formally, $G_i = (\mathcal{S}, \mathcal{A}, \mathcal{T}_i, \mathcal{R}_i)$,

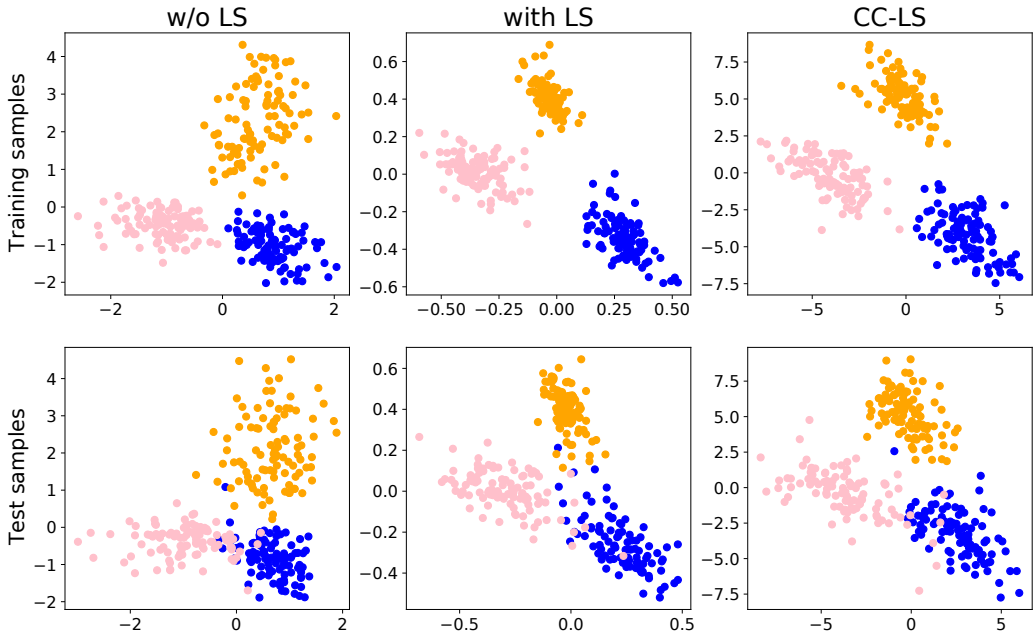


Figure 5.1: Learned representations for the classes “airplane” (blue), “automobile” (orange), and “bird” (pink), projected to an informative 2-dimensional affine subspace spanning the template-vectors of the 3 classes. We show the same plot for samples from the training (above) and test set (below), for the unsmoothed baseline (left), LS (middle), and CC-LS (right), trained with regularization (following the top row of Table 5.1). Note that CC-LS does not concentrate clusters as tightly as LS, suggesting the potential for richer learned representations.

where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}_i(s'|s,a)$ is the probability of transitioning from state s to state s' when executing action a in game G_i, and $\mathcal{R}_i$ is the reward function mapping states and actions to real-valued rewards representing the score of the ith game. In practice, each G_i corresponds to a different videogame from the Atari Learning Environment (Bellemare et al., 2013); each game follows a different set of rules ($\mathcal{T}_i$ and $\mathcal{R}_i$) while taking place on the same console display ($\mathcal{S}$) and controller ($\mathcal{A}$).

In order to train an AMN, we first require access to a set of "experts" $\{E_1, \ldots, E_L\}$, each of which corresponds to an agent specialized in one of the source games. Expert E_i represents a policy π_{E_i} mapping states to distributions over actions, so that we can write $\pi_{E_i}(a_k|s_t)$ for the probability that E_i chooses action $a_k \in \mathcal{A}$ when in state $s_t \in \mathcal{S}$. Note that k indexes the action space, so that $\mathcal{A} = \{a_1, \ldots, a_K\}$, whereas t indexes time (i.e., frame number), so that $\mathcal{S} \supseteq \{s_1, s_2, \ldots\}$. In our implementation, E_i corresponds to a Deep Q-Network (DQN) (Mnih et al., 2015) trained on game G_i, though the fact that E_i is a DQN is not necessary – any policy that performs well on G_i will suffice.

Given the set of expert policies, the AMN is trained to "mimic" the experts in their respective source games. We can reformulate the method as a two-stage process. First, we form an auxiliary dataset of "guidance vectors", $\mathcal{D}_{\text{aux}} = \left\{y_t^{(i)}\right\}$. These vectors are obtained by generating, for each game G_i, a sequence of states $\left\{s_t^{(i)}\right\}_{t=1}^{n}$, and then feeding these states through the corresponding expert policies, in other words:

$$y_t^{(i)} = \left(\pi_{E_i}\left(a_1|s_t^{(i)}\right), \ldots, \pi_{E_i}\left(a_K|s_t^{(i)}\right)\right). \tag{5.9}$$

Second, the parameters of our Actor-Mimic Network, π_θ^{AM}, are learned by minimizing the categorical cross-entropy loss with respect to the auxiliary data:

$$\min_\theta \mathcal{L}^{\text{AMN}}(\theta) = -\sum_{t,i}\sum_k \pi_{E_i}\left(a_k|s_t^{(i)}\right) \cdot \log \pi_\theta^{\text{AM}}\left(a_k|s_t^{(i)}\right). \tag{5.10}$$

In practice, we minimize this loss using minibatch stochastic gradient descent, running the gameplay-

generation in parallel with the gradient steps. The effectiveness of the AMN approach is fundamentally computational; the expert policies can be trained independently in parallel, and the AMN is much faster to optimize via the cross-entropy loss (5.10) than using policy gradients, as it is able to leverage the rich information from the expert policies directly, (with an entire guidance vector of probabilities containing information for all classes at each time step, rather than learning from noisy and biased n-step bootstrap estimates).

Since $y_t^{(i)}$ is a vector of probabilities over actions, our auxiliary data is simplex-valued rather than categorical. It is therefore clear from Equation 5.10 that the AMN model solves a compositional regression problem, whence we propose replacing the cross-entropy loss with its probabilistic counterpart, the continuous categorical log-likelihood:

$$\min_{\theta} \mathcal{L}^{\text{CC-AMN}}(\theta) = -\sum_{t,i} \left\{ \log C\left(\lambda_\theta^{\text{AM}}\left(s_t^{(i)}\right)\right) + \sum_k \pi_{E_i}\left(a_k | s_t^{(i)}\right) \cdot \log \lambda_\theta^{\text{AM}}\left(a_k | s_t^{(i)}\right) \right\}. \quad (5.11)$$

We call this model CC-AMN, and we compare its performance against the AMN model, as well as the DQN baseline.

5.4.1 Experiments

We follow the experimental setup of Parisotto et al (Parisotto, Ba, and Salakhutdinov, 2015), choosing a subset of games from the Atari Learning Environment in which the DQN model performed at super-human level. For each game, we pre-train a DQN with the same network architecture; these are then used as the expert policies. Our network architecture, described in Appendix C.1.2, is taken directly from (Mnih et al., 2015), and is also used for the AMN and CC-AMN models.

First, we reproduce the results of (Parisotto, Ba, and Salakhutdinov, 2015) and compare with our novel CC-AMN model, as shown in Table 5.3. The evaluation scores of the CC-AMN are similar to those of the AMN, except for the game of Pong, where using the CC likelihood leads to unstable training, resulting in worse performance and higher variability in the evaluation score.

Table 5.3: Mean evaluation score (and standard deviation) over the last 20 evaluation epochs (higher is better). With the exception of Pong, the performance of AMN and CC-AMN is similar.

Model	Breakout	Atlantis	Pong	SpaceInvaders
DQN	331 (±44)	32 833 (±14 430)	20.9 (±0.2)	442 (±119)
AMN	337 (±74)	31 558 (±9 084)	20.9 (±0.1)	415 (±126)
CC-AMN	320 (±66)	26 196 (±10 396)	8.8 (±11.9)	415 (±132)

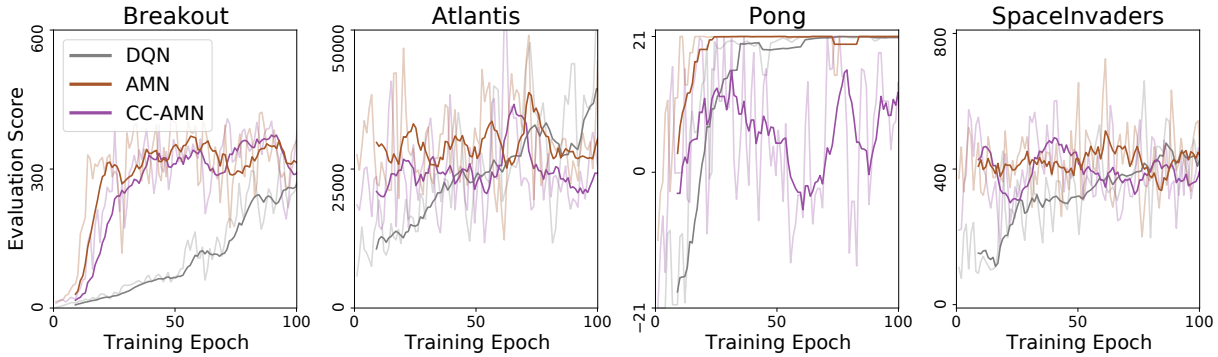


Figure 5.2: Training curves for the CC-AMN and AMN models, run on the simplified single-game objective. Solid lines reflect a moving average of the raw evaluation scores (faded lines). Each training epoch lasts 100 000 frames, with the evaluation scores being calculated from another 100 000 frames. While the actor-mimic models learn much faster than the DQN, the CC-AMN shows no improvement over the AMN model.

Note that both AMN and CC-AMN are generally able to achieve similar performance to the expert DQN.

Second, we focus specifically on the effect of the probabilistic objective (5.11) on network training by reducing the multi-task objective to a single-task objective, i.e., we no longer sum over i in Equation 5.10. This corresponds to running AMN and CC-AMN against the expert DQN, E_i, of a single game, and we do this separately for each game, as shown in Figure 5.2. While both CC-AMN and AMN are able to train much faster than the DQN, converging in just a few epochs, CC-AMN fails to outperform AMN, and can be slower to converge (Breakout) or worse overall (Pong).

The case of Pong highlights an important failure mode of CC-AMN, which also offers some insight as to why our model underperforms in the other games. The issue originates in the normal-

izing constant (5.4), which is numerically unstable when the parameter λ is close to uniform, due to the product of log-ratios vanishing in the in the denominator, as was noted in (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020). In the case of CC-AMN, our optimization hovers around this unstable region, since the guidance vectors tend to concentrate around the centroid of the simplex (this is because our expert policies correspond to the softmax of Q-value functions, which don't typically exhibit large variability across actions since, over small time steps, most actions are not individually critical to the outcome of the game). In practice this is not necessarily a problem, as we zero out the unstable gradients during optimization, as in (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020). Doing so provides a reasonable approximation, since $\nabla G(\lambda) = 0$ for $\lambda = (1/K, \ldots, 1/K)$, by symmetry ($\sum_k \lambda_k = 1$ is constrained by definition of the continuous categorical). Nevertheless, this behavior likely results in a worse optimization landscape overall (unlike in CC-LS where y, and hence λ, are far away from the centroid), which in the game of Pong, derails our gradient search altogether. Further investigation may involve rerunning our experiments with arbitrary-precision floating point, though we currently find this to be computationally prohibitive.

5.5 Discussion and Future Work

Comparing our experiments on CC-LS and CC-AMN, it may come as a surprise that the former yields the more promising empirical results, in spite of its less rigorous theoretical underpinning (with targets that are simplex-valued, but not genuinely continuous). This observation suggests that the continuous categorical may provide useful modeling advances outside the realm of compositional data analysis and probabilistic modeling, for example in classification problems.

It is also worth noting that, throughout our experiments, the cross-entropy loss may have benefitted disproportionately from favorable network initialization, which has been developed for and become increasingly specialized toward networks with particular loss functions (He et al., 2015). A similar argument can be made about network architecture, noting that a good architecture for the cross-entropy loss may not be equivalent to a good architecture for its continuous categorical coun-

terpart. In fact, we have observed experimentally that additional architecture or hyperparameter search can lead to improved performance for the CC-based approaches. However we deliberately chose not to focus on such experimentation, as doing so could further entangle the effect of the loss function on our models; we leave such analyses to future work.

At a more theoretical level, as identified in (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020), it follows from the properties of exponential families that optimizing the continuous categorical log-likelihood results in an unbiased estimator for its mean parameter, which corresponds to a (local) average of the observed data. This is, in fact, akin to the cross-entropy loss, which is also maximized at local average that approximates the conditional expectation of the outputs given the inputs. The optimization landscapes defined by the two loss functions could therefore be of similar nature, though this remains an open question.

Last, we highlight a computational limitation of our approach: the numerical instabilities noted in section 5.4.1 are exacerbated in high dimensions. In fact, evaluating Equation 5.4 for much more than 10 classes is problematic for all $\lambda \in \Delta^K$ (not only for λ around the centroid), since the K summands typically cancel out beyond numerical precision.[2] As a result, we constrained our experimentation to examples where $K \leq 10$, however, similar applications with $K \sim 100$ or greater are also of interest. Further advances in theory or numerical analysis will be needed to enable successful applications of the continuous categorical at this scale.

We conclude by noting that, taken together with the theoretical and empirical results in (Loaiza-Ganem and Cunningham, 2019) and (Gordon-Rodriguez, Loaiza-Ganem, and Cunningham, 2020), our work suggests that future methodological advances may be possible through a combination of careful probabilistic consideration of the cross-entropy loss, and the use of the continuous categorical distribution.

[2]For an intuitive explanation, note that the Lebesgue measure of the K-dimensional simplex is $1/K!$. We therefore expect $C(\lambda)^{-1} \sim 1/K!$. However, as K increases, the "typical" summand in (5.4) decays slower than $1/K!$ (if at all). Thus, for large K, the individual summands in (5.4) will become much larger (in magnitude) than $C(\lambda)^{-1}$. Their summation will then result in (near) total cancellation and therefore total loss of numerical precision.

Conclusion

No busques más que no hay.
Cuando el Rey Don San Fernando
conquistó a Sevilla,
ya se preguntó:
¿Dónde está mi Betis?

Silvio Fernández Melgarejo

To conclude, we summarize the contributions presented in this dissertation, and highlight some directions for future work.

Our initial focus was on predictive models with compositional inputs, in particular, on the problem of selecting sparse, maximally predictive log-ratios. This is a combinatorial optimization problem, on which existing methods struggle to scale up to modern metagenomic datasets. By leveraging continuous relaxations, Chapter 3 showed how to approximate this combinatorial problem with a differentiable objective, which can be optimized efficiently by gradient descent. As a result, we were able to design a learning algorithm that runs several orders of magnitude faster than competing methods, while matching state of the art predictive accuracy and sparsity. As well as evaluating the application of our method on a wide range of microbiome datasets, we provided a self-contained software package implementing our algorithm in R and Python. Our package is currently available on CRAN[3] and Github.[4] In ongoing and future work we further

[3] `https://cran.r-project.org/web/packages/codacore/index.html`
[4] `https://github.com/egr95/py-codacore`

explore the possibilities of our learning algorithm, including novel tasks such as semi-supervised learning and meta-learning.

Our second focus was on predictive models with compositional outputs. To this end, in Chapter 4 we proposed a novel exponential family of probability distributions on the simplex. Our distribution enjoys appealing theoretical properties, with a particularly simple density function whose normalizing constant can be written in closed form in terms of elementary functions. In addition our distribution defines a performant likelihood model for compositional outcomes. This strong performance was demonstrated both on traditional compositional datasets as well as on some noteworthy machine learning applications, such as knowledge distillation. However, one important question remains unsolved: computing the normalizing constant in high dimensions without incurring large floating-point errors. As part of ongoing work, we have made substantive progress towards more general computation methods and analytical techniques for our normalizing constant. However, as of yet, our best efforts fail at around 50 dimensions. Much like the Dirichlet distribution has benefitted from years of progress towards efficient numerical algorithms for computing the Gamma function (Pugh, 2004), we expect that significant progress will be made for our own normalizing constant as well.

More generally, we believe there is still much to be gained by combining techniques and insights from modern machine learning with the principles of compositional data analysis. For example, the jury is still out on whether custom deep learning architectures for CoDa can achieve state of the art predictive performance (Vangay, Hillmann, and Knights, 2019; Quinn et al., 2020). Moreover, highly successful techniques such as data augmentation, self-supervised learning, autoencoders, autoML, hyperparameter optimization, etc., have yet to see significant use in the context of CoDa. As such, we believe the opportunity set at the intersection of ML and CoDa can continue to be a source of great excitement.

References

Aitchison, J and J Bacon-Shone (1999). "Convex linear combinations of compositions". In: *Biometrika* 86.2, pp. 351–364.

Aitchison, John (1982). "The statistical analysis of compositional data". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 44.2, pp. 139–160.

Aitchison, John (1984). "The statistical analysis of geochemical compositions". In: *Journal of the International Association for Mathematical Geology* 16.6, pp. 531–564.

Aitchison, John (1985). "A general class of distributions on the simplex". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 47.1, pp. 136–146.

Aitchison, John (1994). "Principles of Compositional Data Analysis". In: *Lecture Notes-Monograph Series* 24, pp. 73–81.

Aitchison, John (1999). "Logratios and natural laws in compositional data analysis". In: *Mathematical Geology* 31.5, pp. 563–580.

Aitchison, John (2008). "The single principle of compositional data analysis, continuing fallacies, confusions and misunderstandings and some suggested remedies". In: *Proceedings of CoDaWork*.

Aitchison, John and Jim W Kay (2003). "Possible solution of some essential zero problems in compositional data analysis". In: *Proceedings of CoDaWork*.

Alix-Panabières, Catherine and Klaus Pantel (2016). "Clinical applications of circulating tumor cells and circulating tumor DNA as liquid biopsy". In: *Cancer discovery* 6.5, pp. 479–491.

Atchison, J and Sheng M Shen (1980). "Logistic-normal distributions: Some properties and uses". In: *Biometrika* 67.2, pp. 261–272.

Ba, Jimmy and Rich Caruana (2014). "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*, pp. 2654–2662.

Barnard, Kobus et al. (2003). "Matching words and pictures". In: *Journal of machine learning research* 3.Feb, pp. 1107–1135.

Bates, Stephen and Robert Tibshirani (2019). "Log-ratio lasso: Scalable, sparse estimation for log-ratio models". In: *Biometrics* 75.2, pp. 613–624.

Bellemare, Marc G et al. (2013). “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Best, Myron G et al. (2015). “RNA-Seq of tumor-educated platelets enables blood-based pan-cancer, multiclass, and molecular pathway cancer diagnostics”. In: *Cancer cell* 28.5, pp. 666–676.

Bishop, Chris M (1995). “Training with noise is equivalent to Tikhonov regularization”. In: *Neural computation* 7.1, pp. 108–116.

Blei, David M, Andrew Y Ng, and Michael I Jordan (2003). “Latent dirichlet allocation”. In: *Journal of machine Learning research* 3.Jan, pp. 993–1022.

Bondell, Howard D and Brian J Reich (2009). “Simultaneous factor selection and collapsing levels in ANOVA”. In: *Biometrics* 65.1, pp. 169–177.

Breunig, Christian and Marius R Busemeyer (2012). “Fiscal austerity and the trade-off between public investment and social spending”. In: *Journal of European Public Policy* 19.6, pp. 921–938.

Buccianti, A. and E. Grunsky (2014). “Compositional data analysis in geochemistry: Are we sure to see what really occurs during natural processes?” In: *Journal of Geochemical Exploration* 141, pp. 1–5.

Buccianti, Antonella, Glòria Mateu-Figueras, and Vera Pawlowsky-Glahn (2006). “Compositional data analysis in the geosciences: From theory to practice”. In: Geological Society of London.

Buccianti, Antonella and Vera Pawlowsky-Glahn (2005). “New perspectives on water chemistry and compositional data analysis”. In: *Mathematical Geology* 37.7, pp. 703–727.

Buciluǎ, Cristian, Rich Caruana, and Alexandru Niculescu-Mizil (2006). “Model compression”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541.

Calle, M Luz (2019). “Statistical analysis of metagenomics data”. In: *Genomics & informatics* 17.1.

Cammarota, Giovanni et al. (2020). “Gut microbiome, big data and machine learning to promote precision medicine for cancer”. In: *Nature Reviews Gastroenterology & Hepatology* 17.10, pp. 635–648.

Campbell, G and J Mosimann (1987). “Multivariate methods for proportional shape”. In: *ASA Proceedings of the Section on Statistical Graphics*. Vol. 1. Washington, pp. 10–17.

Chen, Guangyong et al. (2019). “Rethinking the Usage of Batch Normalization and Dropout in the Training of Deep Neural Networks”. In: *arXiv preprint arXiv:1905.05928*.

Chorowski, Jan and Navdeep Jaitly (2016). “Towards better decoding and language model integration in sequence to sequence models”. In: *arXiv preprint arXiv:1612.02695*.

Coenders, Germa and Michael Greenacre (2021). “Three approaches to supervised learning for compositional data with pairwise logratios”. In: *arXiv preprint arXiv:2111.08953*.

Connor, Robert J and James E Mosimann (1969). “Concepts of independence for proportions with a generalization of the Dirichlet distribution”. In: *Journal of the American Statistical Association* 64.325, pp. 194–206.

Crovesy, Louise, Daniele Masterson, and Eliane Lopes Rosado (2020). “Profile of the gut microbiota of adults with obesity: a systematic review”. In: *European journal of clinical nutrition* 74.9, pp. 1251–1262.

Demšar, Janez (2006). “Statistical comparisons of classifiers over multiple data sets”. In: *The Journal of Machine Learning Research* 7, pp. 1–30.

Dillies, Marie-Agnès et al. (2013). “A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis”. In: *Briefings in bioinformatics* 14.6, pp. 671–683.

Douma, Jacob C and James T Weedon (2019). “Analysing continuous proportions in ecology and evolution: A practical introduction to beta and Dirichlet regression”. In: *Methods in Ecology and Evolution* 10.9, pp. 1412–1430.

Egozcue, Juan José and Vera Pawlowsky-Glahn (2005). “Groups of parts and their balances in compositional data analysis”. In: *Mathematical Geology* 37.7, pp. 795–828.

Egozcue, Juan José and Vera Pawlowsky-Glahn (2016). “Changing the reference measure in the simplex and its weighting effects”. In: *Austrian Journal of Statistics* 45.4, pp. 25–44.

Egozcue, Juan José et al. (2003). “Isometric logratio transformations for compositional data analysis”. In: *Mathematical Geology* 35.3, pp. 279–300.

Erosheva, Elena A (2002). “Grade of membership and latent structure models with application to disability survey data”. PhD thesis. PhD thesis, Carnegie Mellon University, Department of Statistics.

Fawaz, Hassan Ismail et al. (2019). “Deep learning for time series classification: a review”. In: *Data mining and knowledge discovery* 33.4, pp. 917–963.

Fernandes, Andrew D et al. (2013). "ANOVA-like differential expression (ALDEx) analysis for mixed population RNA-Seq". In: *PLoS One* 8.7, e67019.

Fernandes, Andrew D et al. (2014). "Unifying the analysis of high-throughput sequencing datasets: characterizing RNA-seq, 16S rRNA gene sequencing and selective growth experiments by compositional data analysis". In: *Microbiome* 2.1, p. 15.

Filzmoser, P and B Walczak (2014). "What can go wrong at the data normalization step for identification of biomarkers?" In: *Journal of Chromatography A* 1362, pp. 194–205.

Filzmoser, Peter, Karel Hron, and Clemens Reimann (2009). "Univariate statistical analysis of environmental (compositional) data: problems and possibilities". In: *Science of the Total Environment* 407.23, pp. 6100–6108.

Forslund, Kristoffer et al. (2015). "Disentangling type 2 diabetes and metformin treatment signatures in the human gut microbiota". In: *Nature* 528.7581, pp. 262–266.

Friedman, Jerome, Trevor Hastie, Robert Tibshirani, et al. (2001). *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York.

Fry, Jane M, Tim RL Fry, and Keith R McLaren (2000). "Compositional data analysis and zeros in micro data". In: *Applied Economics* 32.8, pp. 953–959.

Galloway, Angus et al. (2019). "Batch normalization is a cause of adversarial vulnerability". In: *arXiv preprint arXiv:1905.02161*.

Garbin, Christian, Xingquan Zhu, and Oge Marques (2020). "Dropout vs. batch normalization: an empirical study of their impact to deep learning". In: *Multimedia Tools and Applications*, pp. 1–39.

Gertheiss, Jan and Gerhard Tutz (2010). "Sparse modeling of categorial explanatory variables". In: *The Annals of Applied Statistics*, pp. 2150–2180.

Gevers, Dirk et al. (2014). "The treatment-naive microbiome in new-onset Crohn's disease". In: *Cell host & microbe* 15.3, pp. 382–392.

Gilbert, Jack A et al. (2018). "Current understanding of the human microbiome". In: *Nature medicine* 24.4, pp. 392–400.

Gloor, Gregory B and Gregor Reid (2016). "Compositional analysis: a valid approach to analyze microbiome high-throughput sequencing data". In: *Canadian journal of microbiology* 62.8, pp. 692–703.

Gloor, Gregory B et al. (2016). "It's all relative: analyzing microbiome data as compositions". In: *Annals of epidemiology* 26.5, pp. 322–329.

Gloor, Gregory B et al. (2017). “Microbiome datasets are compositional: and this is not optional”. In: *Frontiers in microbiology* 8, p. 2224.

Goodman, Bryce and Seth Flaxman (2017). “European Union regulations on algorithmic decision-making and a “right to explanation””. In: *AI magazine* 38.3, pp. 50–57.

Gordon-Rodriguez, Elliott, Gabriel Loaiza-Ganem, and John Cunningham (2020). “The continuous categorical: a novel simplex-valued exponential family”. In: *International Conference on Machine Learning*. PMLR, pp. 3637–3647.

Gordon-Rodriguez, Elliott, Thomas P Quinn, and John P Cunningham (2022). “Learning sparse log-ratios for high-throughput sequencing data”. In: *Bioinformatics* 38.1, pp. 157–163.

Gordon-Rodriguez, Elliott et al. (2020). “Uses and Abuses of the Cross-Entropy Loss: Case Studies in Modern Deep Learning”. In: *Proceedings on "I Can’t Believe It’s Not Better!" at NeurIPS Workshops*. Vol. 137. Proceedings of Machine Learning Research. PMLR, pp. 1–10.

Greenacre, Michael (2019a). “Comments on: Compositional data: the sample space and its structure”. In: *TEST* 28.3, pp. 644–652.

Greenacre, Michael (2019b). “Variable selection in compositional data analysis using pairwise logratios”. In: *Mathematical Geosciences* 51.5, pp. 649–682.

Greenacre, Michael (2020). “Amalgamations are valid in compositional data analysis, can be used in agglomerative clustering, and their logratios have an inverse transformation”. In: *Applied Computing and Geosciences* 5, p. 100017.

Greenacre, Michael, Eric Grunsky, and John Bacon-Shone (2020). “A comparison of isometric and amalgamation logratio balances in compositional data analysis”. In: *Computers & Geosciences*, p. 104621.

Gueorguieva, Ralitza, Robert Rosenheck, and Daniel Zelterman (2008). “Dirichlet component regression and its applications to psychiatric data”. In: *Computational statistics & data analysis* 52.12, pp. 5344–5355.

He, Haibo and Yunqian Ma (2013). “Imbalanced learning: foundations, algorithms, and applications”. In: pp. 43–59.

He, Kaiming et al. (2015). “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

Hernández-García, Alex and Peter König (2018). “Do deep nets really need weight decay and dropout?” In: *arXiv preprint arXiv:1802.07042*.

Hijazi, Rafiq H and Robert W Jernigan (2009). "Modelling compositional data using Dirichlet regression models". In: *Journal of Applied Probability & Statistics* 4.1, pp. 77–91.

Hijazi, Rafiq Hamed (2003). "Analysis of compositional data using Dirichlet covariate models". PhD thesis. American University.

Hinton, Geoffrey, Oriol Vinyals, and Jeffrey Dean (2015). "Distilling the Knowledge in a Neural Network". In: *NIPS Deep Learning and Representation Learning Workshop*.

Holm, Sture (1979). "A simple sequentially rejective multiple test procedure". In: *Scandinavian journal of statistics*, pp. 65–70.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167*.

Jang, Eric, Shixiang Gu, and Ben Poole (2016). "Categorical reparameterization with gumbel-softmax". In: *arXiv preprint arXiv:1611.01144*.

Katz, Jonathan N and Gary King (1999). "A statistical model for multiparty electoral data". In: *American Political Science Review* 93.1, pp. 15–32.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, Diederik P. and Max Welling (2014). "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Krogh, Anders and John A Hertz (1992). "A simple weight decay can improve generalization". In: *Advances in neural information processing systems*, pp. 950–957.

Li, Hongzhe (2015). "Microbiome, metagenomics, and high-dimensional compositional data analysis". In: *Annual Review of Statistics and Its Application* 2, pp. 73–94.

Li, Xiang et al. (2019). "Understanding the disharmony between dropout and batch normalization by variance shift". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2682–2690.

Linderman, Scott et al. (2018). "Reparameterizing the birkhoff polytope for variational permutation inference". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 1618–1627.

Loaiza-Ganem, Gabriel and John P Cunningham (2019). "The continuous Bernoulli: fixing a pervasive error in variational autoencoders". In: *Advances in Neural Information Processing Systems*, pp. 13266–13276.

Lovell, David et al. (2015). "Proportionality: a valid alternative to correlation for relative data". In: *PLoS Comput Biol* 11.3, e1004075.

Lu, Jiarui, Pixu Shi, and Hongzhe Li (2019). "Generalized linear models with linear constraints for microbiome compositional data". In: *Biometrics* 75.1, pp. 235–244.

Ma, Zhanyu et al. (2016). "Decorrelation of neutral vector variables: Theory and applications". In: *IEEE transactions on neural networks and learning systems* 29.1, pp. 129–143.

Maddison, Chris J, Andriy Mnih, and Yee Whye Teh (2016). "The concrete distribution: A continuous relaxation of discrete random variables". In: *arXiv preprint arXiv:1611.00712*.

Magne, Fabien et al. (2020). "The Firmicutes/Bacteroidetes ratio: a relevant marker of gut dysbiosis in obese patients?" In: *Nutrients* 12.5, p. 1474.

Martín-Fernández, JA, Carles Barceló-Vidal, and Vera Pawlowsky-Glahn (2000). "Zero replacement in compositional data sets". In: *Data analysis, classification, and related methods*. Springer, pp. 155–160.

Martín-Fernández, JA et al. (2018). "Advances in principal balances for compositional data". In: *Mathematical Geosciences* 50.3, pp. 273–298.

Martín-Fernández, Josep Antoni et al. (2012). "Model-based replacement of rounded zeros in compositional data: classical and robust approaches". In: *Computational Statistics & Data Analysis* 56.9, pp. 2688–2704.

Martino, Cameron et al. (2019). "A novel sparse compositional technique reveals microbial perturbations". In: *MSystems* 4.1.

Masoudimansour, Walid and Nizar Bouguila (2017). "Dirichlet Mixture Matching Projection for supervised linear dimensionality reduction of proportional data". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 2806–2810.

Mateu-Figueras, G, C Barceló-Vidal, and V Pawlowsky-Glahn (1998). "Modeling compositional data with multivariate skew-normal distributions". In: *Proceedings of IAMG*. Vol. 98, pp. 532–537.

Mena, Gonzalo E. et al. (2018). "Learning Latent Permutations with Gumbel-Sinkhorn Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Mert, Mehmet Can, Peter Filzmoser, and Karel Hron (2015). "Sparse principal balances". In: *Statistical Modelling* 15.2, pp. 159–174.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533.

Morton, James T et al. (2017). "Balance trees reveal microbial niche differentiation". In: *MSystems* 2.1.

Morton, James T et al. (2019a). "Establishing microbial composition measurement standards with reference frames". In: *Nature communications* 10.1, pp. 1–11.

Morton, James T et al. (2019b). "Learning representations of microbe–metabolite interactions". In: *Nature methods* 16.12, pp. 1306–1314.

Müller, Rafael, Simon Kornblith, and Geoffrey E Hinton (2019). "When does label smoothing help?" In: *Advances in Neural Information Processing Systems*, pp. 4694–4703.

Na, Jong Hyun et al. (2014). "Compositional landscape for glass formation in metal alloys". In: *Proceedings of the National Academy of Sciences* 111.25, pp. 9031–9036.

Naesseth, Christian A. et al. (2017). "Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms." In: *International Conference on Artificial Intelligence and Statistics*, pp. 489–498.

Ng, Kai Wang, Guo-Liang Tian, and Man-Lai Tang (2011). *Dirichlet and related distributions: Theory, methods and applications*. Vol. 888. John Wiley & Sons.

Noguera-Julian, Marc et al. (2016). "Gut microbiota linked to sexual preference and HIV infection". In: *EBioMedicine* 5, pp. 135–146.

Ostner, Johannes, Salomé Carcy, and Christian Lorenz Müller (2021). "tascCODA: Bayesian tree-aggregated analysis of compositional amplicon and single-cell data". In: *Frontiers in genetics*, p. 2334.

Paisley, John W et al. (2010). "A stick-breaking construction of the beta process". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 847–854.

Palarea-Albaladejo, Javier and JA Martín-Fernández (2008). "A modified EM alr-algorithm for replacing rounded zeros in compositional data sets". In: *Computers & Geosciences* 34.8, pp. 902–917.

Parisotto, Emilio, Jimmy Lei Ba, and Ruslan Salakhutdinov (2015). "Actor-mimic: Deep multitask and transfer reinforcement learning". In: *arXiv preprint arXiv:1511.06342*.

Pawlowsky-Glahn, Vera and Antonella Buccianti (2011). *Compositional data analysis*. Wiley Online Library.

Pawlowsky-Glahn, Vera and Juan José Egozcue (2006). "Compositional data and their analysis: an introduction". In: *Geological Society, London, Special Publications* 264.1, pp. 1–10.

Pawlowsky-Glahn, Vera, Juan José Egozcue, and Raimon Tolosana Delgado (2007). "Lecture notes on compositional data analysis". In: *Universitat de Girona*.

Pawlowsky-Glahn, Vera, Juan José Egozcue, and Raimon Tolosana-Delgado (2015). *Modeling and analysis of compositional data*. John Wiley & Sons.

Pawlowsky-Glahn, Vera, Juan José Egozcue, Raimon Tolosana Delgado, et al. (2011). "Principal balances". In: *Proceedings of CoDaWork*, pp. 1–10.

Pawlowsky-Glahn, Vera and Ricardo A Olea (2004). *Geostatistical analysis of compositional data*. Vol. 7. Oxford University Press.

Pearson, Karl (1897). "Mathematical contributions to the theory of evolution.—on a form of spurious correlation which may arise when indices are used in the measurement of organs". In: *Proceedings of the royal society of london* 60.359-367, pp. 489–498.

Potapczynski, Andres, Gabriel Loaiza-Ganem, and John P Cunningham (2020). "Invertible gaussian reparameterization: Revisiting the gumbel-softmax". In: *Advances in Neural Information Processing Systems* 33, pp. 12311–12321.

Prifti, Edi et al. (2020). "Interpretable and accurate prediction models for metagenomics data". In: *GigaScience* 9.3, giaa010.

Pritchard, Jonathan K, Matthew Stephens, and Peter Donnelly (2000). "Inference of population structure using multilocus genotype data". In: *Genetics* 155.2, pp. 945–959.

Pugh, Glendon Ralph (2004). *An analysis of the Lanczos gamma approximation*. University of British Columbia.

Quinn, Thomas P and Ionas Erb (2019). "Using balances to engineer features for the classification of health biomarkers: a new approach to balance selection". In: *bioRxiv*, p. 600122.

Quinn, Thomas P and Ionas Erb (2020). "Amalgams: data-driven amalgamation for the dimensionality reduction of compositional data". In: *NAR Genomics and Bioinformatics* 2.4, lqaa076.

Quinn, Thomas P, Elliott Gordon-Rodriguez, and Ionas Erb (2021). "A Critique of Differential Abundance Analysis, and Advocacy for an Alternative". In: *arXiv preprint arXiv:2104.07266*.

Quinn, Thomas P et al. (2017). "propr: an R-package for identifying proportionally abundant features using compositional data analysis". In: *Scientific reports* 7.1, pp. 1–9.

Quinn, Thomas P et al. (2018). "Understanding sequencing data as compositions: an outlook and review". In: *Bioinformatics* 34.16, pp. 2870–2878.

Quinn, Thomas P et al. (2019). "A field guide for the compositional analysis of any-omics data". In: *GigaScience* 8.9, giz107.

Quinn, Thomas P et al. (2020). "DeepCoDA: personalized interpretability for compositional health". In: *arXiv preprint arXiv:2006.01392*.

Rahat-Rozenbloom, Sari et al. (2014). "Evidence for greater production of colonic short-chain fatty acids in overweight than lean humans". In: *International journal of obesity* 38.12, pp. 1525–1531.

Rayens, William S and Cidambi Srinivasan (1994). "Dependence properties of generalized Liouville distributions on the simplex". In: *Journal of the American Statistical Association* 89.428, pp. 1465–1470.

Real, Esteban et al. (2019). "Regularized evolution for image classifier architecture search". In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33, pp. 4780–4789.

Rivera-Pinto, Javier et al. (2018). "Balances: a new perspective for microbiome analysis". In: *MSystems* 3.4.

Sadowski, Peter and Pierre Baldi (2018). "Neural Network Regression with Beta, Dirichlet, and Dirichlet-Multinomial Outputs". In: OpenReview.net.

Scealy, JL and AH Welsh (2011). "Regression for compositional data by using distributions defined on the hypersphere". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.3, pp. 351–375.

Sheng, Meiling, Zhaohui Dong, and Yanping Xie (2018). "Identification of tumor-educated platelet biomarkers of non-small-cell lung cancer". In: *OncoTargets and therapy* 11, p. 8143.

Shorten, Connor and Taghi M Khoshgoftaar (2019). "A survey on image data augmentation for deep learning". In: *Journal of Big Data* 6.1, p. 60.

Silverman, Justin D et al. (2017). "A phylogenetic transform enhances analysis of compositional microbiota data". In: *Elife* 6, e21887.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Stewart, Connie and Christopher Field (2011). "Managing the essential zeros in quantitative fatty acid signature analysis". In: *Journal of Agricultural, Biological, and Environmental Statistics* 16.1, pp. 45–69.

Stirn, Andrew, Tony Jebara, and David A Knowles (2019). "A New Distribution on the Simplex with Auto-Encoding Applications". In: *arXiv preprint arXiv:1905.12052*.

Susin, Antoni et al. (2020). "Variable selection in microbiome compositional data analysis". In: *NAR Genomics and Bioinformatics* 2.2, lqaa029.

Szegedy, Christian et al. (2016). "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Templ, Matthias (2020). "Artificial Neural Networks to Impute Rounded Zeros in Compositional Data". In: *arXiv preprint arXiv:2012.10300*.

Tolosana-Delgado, R et al. (2019). "On machine learning algorithms and compositional data". In: *Proceedings of the 8th International Workshop on Compositional Data Analysis, Terrassa, Spain*, pp. 3–8.

Tsagris, Michail and Connie Stewart (2018). "A Dirichlet regression model for compositional data with zeros". In: *Lobachevskii Journal of Mathematics* 39.3, pp. 398–412.

Tutz, Gerhard and Jan Gertheiss (2016). "Regularized regression for categorical data". In: *Statistical Modelling* 16.3, pp. 161–200.

Tzeng, Eric et al. (2015). "Simultaneous deep transfer across domains and tasks". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4068–4076.

Uberoi, Elise, Carl Baker, and Richard Cracknell (2019). "General Election 2019: Full Results and Analysis". In: *Parliament UK. July*.

Van Laarhoven, Twan (2017). "L2 regularization versus batch and weight normalization". In: *arXiv preprint arXiv:1706.05350*.

Vangay, Pajau, Benjamin M Hillmann, and Dan Knights (2019). "Microbiome Learning Repo (ML Repo): A public repository of microbiome regression and classification tasks". In: *Gigascience* 8.5, giz042.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Wan, Jonathan CM et al. (2017). "Liquid biopsies come of age: towards implementation of circulating tumour DNA". In: *Nature Reviews Cancer* 17.4, p. 223.

Wang, Hua-Yan et al. (2008). “Dirichlet component analysis: feature extraction for compositional data”. In: *Proceedings of the 25th international conference on Machine learning*. ACM, pp. 1128–1135.

Washburne, Alex D et al. (2017). “Phylogenetic factorization of compositional data yields lineage-level associations in microbiome datasets”. In: *PeerJ* 5, e2969.

Wilcoxon, Frank (1992). “Individual comparisons by ranking methods”. In: *Breakthroughs in statistics*. Springer, pp. 196–202.

Zhang, Yu-Hang et al. (2017). “Identifying and analyzing different cancer subtypes using RNA-seq data of blood platelets”. In: *Oncotarget* 8.50, p. 87494.

Zoph, Barret et al. (2018). “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710.

Appendix A: Learning Sparse Log-Ratios

A.1 Additional Methodological Detail

A.1.1 Continuous Relaxation

We optimized Eq. 3.8 from the main text using gradient descent with the following specifications:

- **Initialization.** We initialized the weights parameter $\mathbf{w}$ at zero, which corresponds to assigning no prior "preference" to one input variable over any another. As for the intercept parameter α, initialization requires some care, particularly in order to account for possibly imbalanced classes and for the stage-wise additive nature of our model. Namely, α should captures the central tendency of the response so that the gradients propagated to $\mathbf{w}$ capture the *differential* effect of the input variables on the response (not doing so severely hinders gradient descent). In practice, this amounts to initializing α at the estimated coefficient of a generalized linear model with nothing but an intercept (which, in the regression case, reduces to the mean of the response). Note that, from the second stage of ensembling onwards, the fitted value of the current ensemble must be incorporated into said generalized linear model as a fixed offset term. Finally, we found the initialization of β to matter less, and simply used the initial value $\beta = 0.1$ throughout.

- **Epochs.** We trained each continuous relaxation for 100 epochs. While this number of epochs was sufficient for the linear functionals we considered, we expect a higher number would be needed in order to combine our balances (or amalgamations) with nonlinear functionals.

- **Learning rate.** We used an adaptive scheme that exploits the structure of our continuous relaxation. Intuitively, we seek a learning rate that will allow $\mathbf{w}$ to explore the range of

values where our relaxation has non-vanishing gradients, which approximately corresponds to $w_i \in (-5, 5)$. This interval corresponds to soft assignments approximately between $\tilde{w}_i \in (-0.99, 0.99)$; outside of this range, the sigmoid starts to saturate and gradients vanish. The question is then: how to choose a learning rate that allows $\mathbf{w}$ to explore the range $(-5, 5)$ efficiently (i.e., over a minimal number of epochs). We solve this indirectly; we pick the unique learning rate that would result in an initial gradient step that takes the maximum weight, $\max(|\mathbf{w}|)$, to the value 0.5. This learning rate can be found by simply computing one backward pass (and taking the maximum over the gradients w.r.t. $\mathbf{w}$) prior to applying any gradient steps. Note that the value of 0.5 is somewhat arbitrary, and simply represents the size of an initial step that should be "small, but significant relative to the target range of $(-5, 5)$".

- **Momentum.** We used momentum, with rate 0.9.

- **Minibatching.** We did not use minibatching, as we found it unnecessary on the datasets we considered. Nevertheless, minibatching could be beneficial, particularly for datasets with larger number of observations.

A.1.2 Discretization

Given a trained vector of soft assignments $\tilde{\mathbf{w}}$, typically only a small number will have converged to values close to $+1$ or -1, with most components remaining close to zero. As discussed in Section 3.3.3, in order to identify a balance it is sufficient to specify a threshold value t. However, different thresholds $t \in (0, 1)$ can result in different balances, depending on the values of $\tilde{\mathbf{w}}$. Our implementation evaluates a set of 20 "candidate thresholds", $t \in \{t_1, \ldots, t_{20}\}$, and settles on the one that yields the best cross-validation score. Any grid of values $\{t_1, \ldots, t_{20}\} \in (0, 1)$ can be used, where we write $\{t_1, \ldots, t_{20}\}$ in decreasing order. A principled choice is to "step through" the weights. Namely, start by rescaling the positive and negative components of $\tilde{\mathbf{w}}$ so that $\max \tilde{\mathbf{w}}^+ = \max \tilde{\mathbf{w}}^-$, and then set $\{t_1, \ldots, t_{20}\}$ to equal the top 20 order statistics of the set $\{|\tilde{w}_j|\}$. Thus, the

first candidate t_1 will yield a simple log-ratio between two input variables (the rescaling ensure precisely two of our weights are $\geq t_1$), the second candidate t_2 will yield a log-ratio with a total of three input variables, and so forth.

Given a set of candidate thresholds $\{t_1, \ldots, t_{20}\}$, we associate a score to each of these via cross-validation. We first split the training data into 5 folds (sampled with stratification by case control (He and Ma, 2013)). Then, for each candidate threshold t_i, we fit 5 regressors of the form Eq. 3.3 on the 5 cross-validation training sets, we evaluate on their respective validation folds, and average the results to obtain the overall cross-validation score of t_i. Finally, we choose the largest threshold (i.e., the sparsest model) whose cross-validation score is within 1 standard error of the optimum achieved over $\{t_1, \ldots, t_{20}\}$ (with standard errors computed over the 5 cross-validation folds).

Note that we could search in this way over more than 20 candidate thresholds; the choice of the number 20 is by analogy to *selbal* (Rivera-Pinto et al., 2018), where balances of up to (but no more than) 20 input variables are considered. We also found that our results were broadly insensitive to searching over more candidate thresholds; increasing this number can lead to slightly more accurate models that are slightly less sparse, and, evidently, somewhat slower to compute.

A.2 Datasets

Table A.3 provides further details on the 25 datasets used in Section 4.5 of the the main text. Note also that zero-replacement is necessary prior to applying log-ratio transformations, a standard pre-processing step in the field of CoDa (Martín-Fernández, Barceló-Vidal, and Pawlowsky-Glahn, 2000; Aitchison and Kay, 2003; Martín-Fernández et al., 2012). When necessary, we carried out zero-replacement by simply adding one unit to all counts prior to normalization.

A.3 Additional Experimental Detail

Tables A.4, A.5, A.6, A.8 show our evaluation metrics on each individual dataset for a selection of our models. Table 3.2 from the main text summarizes these four tables by averaging over all datasets. Figure A.1 shows the same heatmap as Figure 3.2 from the main text, but with amalga-

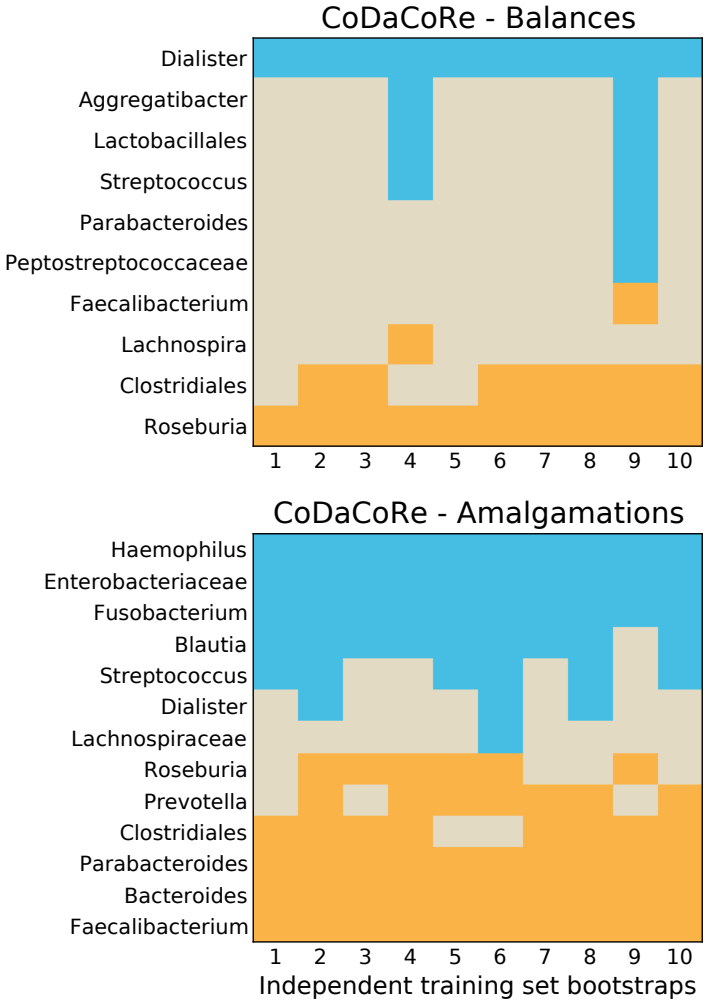


Figure A.1: CoDaCoRe variable selection for the first (most explanatory) log-ratio on the Crohn disease data (Rivera-Pinto et al., 2018). For each of 10 independent bootstraps of the training set (80% of the data randomly sampled with stratification by case-control), we show which variables are selected in the numerator (blue) and denominator (orange) of the log-ratio. Both versions of CoDaCoRe, with balances (top) or amalgamations (bottom), learn remarkably consistent log-ratios across independent training sets. Differences between the sets selected by CoDaCoRe with balances vs. CoDaCoRe with amalgamations can be explained by differences in how the geometric mean vs. summation operations impact the log-ratio. The geometric mean, being more sensitive to small numbers, is more affected by the presence of rarer bacteria species like Dialister and Roseburia (as compared with the more common bacteria species like Haemophilus and Faecalibacterium).

mations as well as balances. Differences between the sets selected by CoDaCoRe with balances vs. CoDaCoRe with amalgamations can be explained by differences in how the geometric mean vs. summation operations impact the log-ratio. The geometric mean, being more sensitive to small numbers, is more affected by the presence of rarer bacteria species like Dialister and Roseburia (as compared with the more common bacteria species like Haemophilus and Faecalibacterium).

A.3.1 Critical Difference Diagrams

Critical difference (CD) diagrams are a robust non-parametric tool for the comparison of multiple classifiers over multiple datasets (Demšar, 2006). For a given success metric (e.g., prediction error), these diagrams display the average rank of each method, taken by sorting the methods from best to worst on each individual dataset. In addition, horizontal lines link together the methods whose ranks were not significantly different from one another, for the metric under consideration. We measure significance at the 5% level, using the Wilcoxon signed rank test (Wilcoxon, 1992) with Holm's alpha correction (Holm, 1979).[1] Figure A.2 shows the critical difference diagrams evaluated over our 6 interpretable CoDa methods: selbal, amalgam, pairwise log-ratios, Coda-lasso, and CoDaCoRe (with default parameters). We show a separate diagram for each of our 5 performance metrics: runtime, sparsity (proportion of variables active), classification accuracy, AUC, and F1 score (all measured on the held-out test set). The results are broadly consistent with Table 3.2 from the main text. Namely, CoDaCoRe performs highly across all of our criteria; it is (by far) the most computationally efficient, as well as the sparsest (tied with selbal and pairwise log-ratios), and the most predictive (tied with amalgam in terms of accuracy, and tied with most other competitors in AUC and F1 score). Note that while CoDaCoRe (with default parameters) scored equally well on AUC and F1 relative to other methods, when running CoDaCoRe with reduced regularization ($\lambda = 0$), it did become significantly more predictive than any of these methods (at the expense of some sparsity). Figure A.3 corroborates these observations, showing the direct pairwise comparisons between each method, separately for each evaluation metric.

[1] We used the implementation of Fawaz et al. (2019) available at https://github.com/hfawaz/cd-diagram

A.3.2 Ablations

While CoDaCoRe clearly outperforms its competitors in terms of runtime and sparsity, the differences in predictive accuracy are more nuanced, and often not statistically significant. One may therefore ask whether there exist classes of problems on which CoDaCoRe will tend to out- or under-perform competing methods. To answer this question, we regress the rank scores from the previous section against (a) number of observations, (b) number of input variables, (c) class imbalance, and (d) a proxy for the signal-to-noise ratio. Namely, for each of our 25 datasets we first rank the predictive accuracy score of CoDaCoRe relative to the 5 other methods (lower rank is better), resulting in 25 datapoints. We then plot these separately against each of (a), (b), (c), and (d), together with the corresponding trendline (obtained from 4 separate univariate regressions). The class imbalance (c) is shown as a number between 0.5 and 1, representing the proportion of datapoints in the majority class (i.e., the baseline accuracy). The signal-to-noise ratio (d) was approximated by taking the out-of-sample accuracy of the Random Forest model, and subtracting the baseline accuracy (majority vote).[2] The results are shown in Figure A.4. While our collection of 25 datasets may not be sufficient to draw firm conclusions, it seems that CoDaCoRe performs consistently across high- and low-dimensional datasets, as well as across the high- and low-signal datasets. Interestingly, CoDaCoRe tends to outperform most strongly on the datasets where the number of observations is small, possibly indicating superior sample efficiency over competing methods. Plotting a similar ablation against the ratio n/p shows a very similar effect, and is omitted for brevity. CoDaCoRe also tends to rank slightly higher on datasets with lower class imbalance, indicating that our algorithm may benefit from class rebalancing, a line that is left to future work.

A.3.3 Implementation Detail for Competing Methods

The competing methods used in our experiments were ran as follows:

- Selbal (Rivera-Pinto et al., 2018) was run using the `selbal.cv` function from the imple-

[2]Note this metric is intended as a rough proxy and we do not claim to accurately estimate the signal-to-noise ratio of any of our datasets.

mentation of the original authors, with default parameters.[3]

- Pairwise log-ratios (Greenacre, 2019b) was run using the `pra` function from the `propr` package (available on CRAN), with `ndim=10`.

- Lasso was run using `cv.glmnet` from the `glmnet` package, with default parameters. The regularization strength was selected using the 1-standard-error rule.

- Coda-lasso (Lu, Shi, and Li, 2019) was run using the implementation of Susin et al. (2020).[4] In particular, we combined the `coda_logistic_lasso` function with cross-validation (again, with the 1-standard error rule), to select the regularization strength.

- Amalgam (Quinn and Erb, 2020) was run using the `amalgam` function from the implementation of the original authors.[5] Since our work compares log-ratio selection methods, we used the parameter `asSLR=TRUE`, which in turn implied we had to change the `numAmal` parameter from its default value of 3 to 6, since an even number is required (for the numerators and denominators of the SLR, respectively).

- DeepCoDA (Quinn et al., 2020) was adapted from the implementation of the original authors (in order to run their Tensorflow code in R).[6]

- CLR-lasso (Susin et al., 2020) was implemented by applying `cv.glmnet` to CLR-transformed data, again, choosing the regularization parameter with the 1-standard-error rule.

- Random Forest was run with the `tuneRF` function from the `randomForest` package (available on CRAN), with `doBest=TRUE` (and default parameters otherwise). This is in order to select the best value of the `mtry` parameter with respect to out-of-bag error.

- Log-ratio lasso (Bates and Tibshirani, 2019) was run using the `cv_two_stage` function

[3] https://github.com/malucalle/selbal
[4] https://github.com/malucalle/CoDA-Penalized-Regression
[5] https://github.com/tpq/amalgam
[6] https://github.com/nphdang/DeepCoDA

from the `logratiolasso` package written by the original authors, with default parameters.[7]

A.4 Simulation Study

A.4.1 Low-Dimensions

Our first simulated dataset is taken directly from Susin et al. (2020). The data-generating process starts with the observed HTS counts of the Crohn's Disease data from Gevers et al. (2014). These data consist of $n = 975$ samples and $p = 48$ genera. For each simulation scenario, the abundance table was obtained by randomly selecting k "active" variables and $\tilde{k}$ "inactive" variables from the p columns, and discarding the rest. The simulation parameters were varied over $k \in \{3, 5, 10\}$ and $\tilde{k} \in \{10, 20, 30, 40\}$. For each such combination, 10 independent simulations were generated, and a balance was defined over the k active variables. For example, in the case $k = 3$, the first variable was used for the numerator, and the other 2 for the denominator.[8] In turn, a binary response was simulated for each datapoint by drawing a Bernoulli random variable, with probability of success equal to the sigmoid of the computed balance. In all, this gave a total of $3 \times 4 \times 10 = 120$ simulated datasets, on each of which we fitted CoDaCoRe (with default parameters), selbal (Rivera-Pinto et al., 2018), and coda-lasso (Lu, Shi, and Li, 2019). The results are shown in figure A.5. Perhaps unsurprisingly (given the large number of samples relative to the number of input variables), all three methods are able to recover the true active variables in the data-generating process, across all 120 datasets. Both CoDaCoRe and selbal were able to do so while maintaining a very low false positive rate, meaning that they almost never selected any of the variables that were not part of the true data-generating process. On the other hand, coda-lasso selected a large amount of false positive variables on many of the datasets, in spite of having tuned its regularization parameter (i.e., L_1 penalty) using the sparsity-inducing 1-standard-error rule on each individual dataset.

[7] https://github.com/stephenbates19/logratiolasso

[8] Note that in the $k = 10$ case, Susin et al. (2020) used an imbalanced log-linear combination, which we changed to a regular balance for the purposes of our study.

A.4.2 Heterogeneous balances

We additionally evaluate the ability of CoDaCoRe to recover *multiple* log-ratios, in the more challenging setting where the true data-generating process is driven by two balances with heterogeneous effect sizes. To do so, we again follow the simulation scheme of Susin et al. (2020), as per the previous section, except for now taking $k = 6$ active variables. Of the 6 variables active, 3 will again form one balance (with the first variable in the numerator and the other 2 in the denominator), and the other 3 will form a "secondary" balance (also with 1 numerator and 2 denominator variables). The effect size of the secondary balance is set to 1/3 of that of the first balance. In other words, the binary response is simulated by computing a linear combination of two balances, the sigmoid of which equals the probability of success. We repeat the process over the same previous values of $\tilde{k} \in \{10, 20, 30, 40\}$, with 10 independent draws for each. On the resulting 40 datasets, we fit CoDaCoRe, selbal and coda-lasso. CoDaCoRe will automatically find two log-ratios (provided there exists sufficient signal in the data), but to accomplish the same with selbal, we had to run the algorithm twice, feeding the first balance as a covariate in the second pass. On the other hand, coda-lasso does not establish balances per se, but rather a log-linear combination where each input variable has a different weight. While it is not obvious in general how these may be combined to produce bona fide balances, in this case we simply measured the ability of coda-lasso to recover all 6 active variables. Note that this is a substantially less stringent criteria than was demanded of CoDaCoRe or selbal, where we compared the first learned balance specifically to the 3 variables from the primary (true) balance, and the second learned balance to the 3 variables from the secondary (true) balance. The results are shown in Table A.1. CoDaCoRe is the only method that shows appreciable performance at recovering the secondary balance, typically selecting at least 2 of its 3 true constituents, while keeping a low false positive rate (8%, on average). Selbal is unable to identify the secondary balance in this heterogeneous setting, while coda-lasso does identify all the relevant variables, but with a problematic false positive rate of 38%, on average (and without specifying how the selected variables may be grouped into balances).

Table A.1: Variable selection performance for CoDaCoRe, selbal, and coda-lasso for simulated datasets with two generative balances of heterogeneous effect sizes. In all, $4 \times 10 = 40$ datasets were used to produce this table. For CoDaCoRe and selbal, we can distinguish between the first and the second learned log-ratios, whereas coda-lasso learns a single log-linear combination for the entire dataset.

	TPR	FPR
CoDaCoRe (first learned balance)	1.00 ± 0.00	0.01 ± 0.02
CoDaCoRe (second learned balance)	0.61 ± 0.29	0.08 ± 0.11
selbal (first learned balance)	1.00 ± 0.00	0.00 ± 0.00
selbal (second learned balance)	0.00 ± 0.00	0.13 ± 0.07
coda-lasso	1.00 ± 0.00	0.38 ± 0.11

A.4.3 High-Dimensions

Next, we consider a collection of purely simulated datasets of arbitrary sizes, which will allow us to evaluate the variable selection capabilities of our methods across a wide range of input dimensions. To simulate the HTS count data of size $n \times p$, we start by drawing $\alpha \sim \text{Dirichlet}(1/\log(p), \ldots, 1/\log(p))$.[9] Then, for each datapoint $i = 1, \ldots, n$, we draw $p_i \sim \text{Dirichlet}(\alpha)$ and $n_i \sim \text{Poisson}(1000 \cdot n)$, and in turn $x_i \sim \text{Multinomial}(n_i, p_i)$. In addition, we corrupt a randomly selected 0.1% of the entries of each x_i with random Poisson(1000) noise. A balance or a summed-log-ratio is then constructed by taking $k/2$ variables in the numerator and $k/2$ variables in the denominator, and its sigmoid defines the probability of success of a Bernoulli binary response. For each $p \in \{100, 300, 1\,000, 3\,000, 10\,000\}$, and $k \in \{2, 4, 8, 16, 32\}$, we simulate 10 independent datasets, resulting in $5 \times 5 \times 10 = 250$ datasets. For each of these 250 datasets, we generate two versions, one where the response is simulated using balances, and one where the response is simulated using summed-log-ratios. We take $n = 200$ throughout. On the datasets simulated using balances, we compare CoDaCoRe (with default parameters) against selbal and coda-lasso, as shown in Figure A.6. On those simulated using summed-log-ratios, we compare CoDaCoRe (with amalgamations) against amalgam (Quinn and Erb, 2020), as shown in Figure A.7. Over-

[9] The parameter $1/\log(p)$ was chosen to ensure the count table obtained downstream is sparse and heavy-tailed.

all, we find that CoDaCoRe performs competitively, selecting most of the relevant input variables even on challenging high-dimensional datasets, all while maintaining low false positive rates. CoDaCoRe recovered the true balance slightly more accurately than selbal and coda-lasso, although with slightly worse false positive rates. The outperformance of CoDaCoRe was most pronounced in datasets with high p or high k. Importantly, selbal failed to terminate (within 100 hours) on the high-dimensional datasets; CoDaCoRe ran in a few seconds on all datasets. Likewise, in the summed-log-ratio case, CoDaCoRe clearly outperformed amalgam, with slightly worse true positive rates but dramatically better false positive rates across the board. Importantly, amalgam also failed to terminate (within 100 hours) on the high-dimensional datasets, in which CoDaCoRe was still able to run in a matter of seconds.

The outperformance of CoDaCoRe on datasets with high k showcases one of the strengths of our gradient descent procedure; since balances are learned "jointly" over all input variables, we are able to more accurately recover balances that involve a larger number of input variables (a hard problem in general). This is in contrast to the stepwise procedure used in selbal, which is more effective at identifying balances that involve a small number of input variables (i.e., those that require a small number of correct "steps" from initialization), but struggles to recover balances over many input variables (since they require a large number of correct "steps"). As the number of steps required to find a balance increases, the likelihood of making an error along the way also increases. This issue does not afflict CoDaCoRe, since gradient descent computes the contribution of each input variable to the learned balance *jointly*, "pushing" the most relevant ones towards being selected in the discretization step downstream.

Table A.2: Data description. n denotes the number of observations, p the number of input variables. We also show the number of observations in the case and control groups.

Dataset ID	n	p	Group 1	Group 2
1	975	48	Crohn's disease	Without
2	128	60	Men who have sex with men	Without
3	220	153	Control	IBD
4	164	158	Crohn's disease	Ulcerative colitis
5	220	885	Control	IBD
6	164	885	Crohn's disease	Ulcerative colitis
7	182	278	Case	Diarrheal control
8	247	610	Case	Non-Diarrheal control
9	292	1133	Colorectal cancer (CRC)	Without
10	318	1302	Colorectal cancer (CRC)	Non-CRC control
11	1182	188	Primary solid tumor	Solid tissue normal
12	1004	188	Her2 Cancer	Not Her2 Cancer
13	718	188	LumA Cancer	LumB Cancer
14	140	992	Crohn's disease (ileum)	Without (ileum)
15	160	992	Crohn's disease (rectum)	Without (rectum)
16	2070	3090	GI tract	Oral
17	180	3090	Female	Male
18	404	3090	Stool	Tongue (dorsum)
19	408	3090	Subgingival plaque	Supragingival plaque
20	172	980	Healthy	Colorectal cancer
21	124	2526	Without	Diabetes
22	130	2579	Cirrhosis	Without
23	199	660	Black	Hispanic
24	342	660	Nugent score high	Nugent score low
25	200	660	Black	White

Table A.3: Data description.

Dataset ID	Group 1 Size	Group 2 Size	Source
1	662	313	doi: 10.1016/j.chom.2014.02.005
2	73	55	doi: 10.1016/j.ebiom.2016.01.032
3	56	164	doi: 10.1038/s41564-018-0306-4
4	88	76	doi: 10.1038/s41564-018-0306-4
5	56	164	doi: 10.1038/s41564-018-0306-4
6	88	76	doi: 10.1038/s41564-018-0306-4
7	93	89	doi: 10.1128/mBio.01021-14
8	93	154	doi: 10.1128/mBio.01021-15
9	120	172	doi: 10.1186/s13073-016-0290-3
10	120	198	doi: 10.1186/s13073-016-0290-3
11	1078	104	doi: 10.1038/ng.2764
12	77	927	doi: 10.1038/ng.2764
13	524	194	doi: 10.1038/ng.2764
14	78	62	doi: 10.1016/j.chom.2014.02.005
15	68	92	doi: 10.1016/j.chom.2014.02.005
16	227	1843	doi: 10.1038/nature11209
17	82	98	doi: 10.1038/nature11209
18	204	200	doi: 10.1038/nature11209
19	203	205	doi: 10.1038/nature11209
20	86	86	doi: 10.1101/gr.126573.111
21	59	65	doi: 10.1038/nature11450
22	68	62	doi: 10.1038/nature13568
23	104	95	doi: 10.1073/pnas.1002611107
24	97	245	doi: 10.1073/pnas.1002611107
25	104	96	doi: 10.1073/pnas.1002611107

Table A.4: Average runtime over 20 train/test splits, in seconds.

Dataset ID	CoDaCoRe (Defaults)	selbal	Pairwise log-ratios	Coda-lasso	amalgam	Random Forest
1	6	328	17	39	48	2
2	5	354	7	19	40	0
3	5	1307	50	62	145	1
4	4	1291	28	44	212	1
5	4	31407	638	420	2553	5
6	4	26638	387	348	3806	4
7	4	3316	79	39	532	1
8	4	14058	522	157	1965	3
9	4	48394	985	391	4694	11
10	5	60138	1540	505	5300	15
11	5	1954	240	90	315	5
12	6	2290	144	59	277	6
13	5	2014	117	107	307	5
14	4	33619	961	99	2919	2
15	5	35193	1326	116	2957	3
16	6	279072	298672	7776	40949	135
17	5	300431	10449	1628	22777	10
18	3	322726	9997	3131	25130	8
19	6	325642	24691	2890	24908	25
20	4	33150	468	208	2822	3
21	4	196216	1036	2942	15328	6
22	4	208464	769	4668	20612	6
23	4	15818	530	84	1447	2
24	4	16382	969	139	1621	4
25	4	15640	553	113	2349	2
Mean	5	79034	14207	1043	7361	11

Table A.5: Proportion of input variables active (%), averaged over 20 train/test splits

Dataset ID	CoDaCoRe (Defaults)	selbal	Pairwise log-ratios	Coda-lasso	amalgam	Random Forest
1	15.4±2.4	25.6±3.1	21.0±2.4	53.8±5.5	85.2±2.8	100.0±0.0
2	4.5±0.6	5.5±0.4	11.2±1.9	9.4±0.7	86.1±2.2	100.0±0.0
3	2.1±0.3	3.0±0.5	3.5±0.9	6.7±1.6	80.7±2.7	100.0±0.0
4	2.0±0.3	1.8±0.2	4.2±0.8	6.3±1.4	81.6±1.6	100.0±0.0
5	0.6±0.3	0.2±0.0	0.7±0.1	1.9±0.7	84.9±0.9	100.0±0.0
6	0.2±0.0	0.3±0.0	0.8±0.1	0.7±0.1	85.3±0.8	100.0±0.0
7	3.3±0.6	0.8±0.1	0.2±0.1	2.2±0.5	79.6±2.5	100.0±0.0
8	1.3±0.2	1.0±0.1	1.7±0.2	4.2±1.0	81.9±2.2	100.0±0.0
9	0.8±0.1	0.2±0.0	0.2±0.1	1.2±0.4	89.7±1.0	100.0±0.0
10	0.6±0.1	0.2±0.0	0.0±0.0	1.1±0.2	90.0±1.3	100.0±0.0
11	1.8±0.9	4.2±0.5	2.5±0.3	74.2±10.6	91.0±1.7	100.0±0.0
12	5.3±0.9	6.9±0.5	5.0±0.9	62.1±8.8	89.0±1.5	100.0±0.0
13	5.2±0.4	7.1±0.4	5.9±0.8	58.1±12.7	84.5±1.7	100.0±0.0
14	0.8±0.2	0.2±0.0	0.6±0.1	0.4±0.1	92.9±1.4	100.0±0.0
15	0.6±0.2	0.3±0.1	0.6±0.1	0.6±0.2	90.8±1.1	100.0±0.0
16	0.1±0.0	0.4±0.0	0.2±0.0	95.1±9.9	94.8±1.3	100.0±0.0
17	0.1±0.0	0.1±0.0	0.0±0.0	0.0±0.0	94.0±2.6	100.0±0.0
18	0.1±0.0	0.3±0.0	0.3±0.0	100.0±0.0	88.6±4.5	100.0±0.0
19	0.1±0.0	0.1±0.0	0.3±0.0	0.4±0.1	97.8±1.1	100.0±0.0
20	0.5±0.1	0.2±0.0	0.4±0.1	0.4±0.5	89.1±2.5	100.0±0.0
21	0.3±0.1	0.1±0.0	0.0±0.0	1.1±0.9	92.1±3.7	100.0±0.0
22	0.1±0.0	0.1±0.0	0.3±0.1	10.0±10.7	89.9±1.8	100.0±0.0
23	0.5±0.1	0.3±0.0	0.2±0.1	0.5±0.3	86.7±3.3	100.0±0.0
24	1.0±0.1	0.8±0.1	1.9±0.1	1.6±0.1	93.2±1.2	100.0±0.0
25	0.3±0.0	0.3±0.0	0.7±0.1	0.2±0.1	69.8±5.7	100.0±0.0
Mean	1.9±0.3	2.4±0.2	2.5±0.4	19.7±2.7	87.6±2.1	100.0±0.0

Table A.6: Out-of-sample accuracy (%) averaged over 20 train/test splits.

Dataset ID	CoDaCoRe (Defaults)	selbal	Pairwise log-ratios	Coda-lasso	amalgam	Random Forest
1	71.4±1.1	45.4±2.9	68.5±1.0	72.4±1.7	76.1±1.4	81.5±1.4
2	87.5±2.5	60.6±3.6	88.3±3.1	88.7±2.8	89.4±2.8	90.8±2.3
3	76.7±2.4	73.3±0.0	73.4±0.2	66.0±3.0	73.7±2.4	81.7±1.5
4	61.9±2.6	52.6±1.9	70.0±2.7	64.8±3.0	67.8±3.7	75.0±2.4
5	85.0±2.0	73.4±0.2	82.4±1.5	76.6±3.4	82.7±2.4	87.9±2.0
6	65.3±3.5	55.1±2.0	68.2±3.0	67.3±3.0	63.1±2.9	73.2±3.1
7	69.0±3.8	64.0±2.8	53.0±2.0	66.9±4.0	60.4±2.9	69.7±5.6
8	92.9±1.4	69.2±1.8	88.6±2.3	93.1±1.4	92.1±1.5	96.2±1.0
9	61.4±2.9	64.2±1.4	59.1±0.3	59.1±3.9	65.7±3.2	67.5±2.1
10	63.1±1.6	66.0±0.7	62.5±0.0	65.9±3.3	63.2±2.7	66.4±1.1
11	96.8±0.5	68.1±9.2	98.9±0.3	97.3±0.6	99.1±0.2	99.2±0.2
12	92.9±0.5	92.1±0.0	92.2±0.2	85.8±1.0	92.6±0.6	92.4±0.2
13	79.1±1.0	27.2±0.1	75.5±0.7	80.5±1.2	82.6±1.0	83.7±1.2
14	68.5±3.8	50.3±1.8	68.1±3.8	62.2±4.0	65.0±3.5	63.8±4.3
15	74.5±4.9	57.9±1.8	72.0±3.3	71.5±4.8	65.8±3.2	70.8±5.0
16	99.9±0.1	88.9±0.0	100.0±0.0	99.9±0.1	99.9±0.1	99.9±0.1
17	51.4±3.3	54.8±1.7	53.5±0.7	46.0±0.0	51.9±4.0	61.2±2.9
18	100.0±0.0	49.9±0.5	100.0±0.0	100.0±0.0	99.5±0.5	99.8±0.4
19	64.7±2.5	55.2±1.5	65.2±2.0	70.5±1.8	70.8±1.6	73.2±2.1
20	69.0±4.1	62.4±2.2	66.7±3.6	52.6±2.4	60.6±3.6	68.3±3.0
21	60.8±4.1	54.6±2.6	52.2±0.4	54.8±4.9	57.4±6.1	54.4±5.4
22	77.2±3.2	64.6±4.0	75.9±5.5	72.9±3.9	74.1±3.6	88.0±2.1
23	59.0±3.4	48.9±1.5	52.0±1.8	52.5±1.3	56.2±3.0	54.0±3.0
24	93.3±1.4	76.5±1.6	92.9±1.3	93.8±1.3	93.6±0.8	93.9±1.1
25	59.6±3.1	53.9±1.7	53.7±2.9	51.2±0.0	57.7±3.7	56.5±2.4
Mean	75.2±2.4	61.2±1.9	73.3±1.7	72.5±2.3	74.4±2.5	78.0±2.2

Table A.7: Out-of-sample AUC (%) averaged over 20 train/test splits.

Dataset ID	CoDaCoRe (Defaults)	selbal	Pairwise log-ratios	Coda-lasso	amalgam	Random Forest
1	75.8±1.7	81.5±1.5	69.9±2.5	81.2±1.8	81.0±1.5	87.8±1.2
2	93.9±1.9	92.3±2.1	94.2±2.0	94.8±1.7	94.7±2.1	96.2±2.0
3	79.2±2.9	76.0±3.2	77.3±3.2	73.9±5.5	76.9±3.9	88.1±1.6
4	69.4±3.3	73.4±4.5	74.3±3.0	74.0±3.3	72.6±3.4	83.2±2.8
5	91.2±1.6	93.1±1.1	90.0±1.8	92.1±1.8	88.8±2.8	93.7±1.3
6	72.0±3.7	74.9±4.1	76.2±3.7	76.0±3.7	68.0±3.5	81.8±3.3
7	73.7±4.0	82.1±3.0	53.0±2.9	75.6±4.9	65.8±3.5	75.3±4.7
8	98.2±0.7	96.5±0.9	94.8±2.0	98.8±0.5	97.6±0.8	99.3±0.3
9	64.6±3.0	61.6±3.5	51.5±1.6	61.9±4.3	67.7±3.5	70.5±2.1
10	66.2±3.7	62.8±2.0	50.0±0.0	68.5±2.7	64.3±3.1	66.0±2.0
11	98.5±0.5	99.6±0.3	99.5±0.4	99.4±0.6	99.4±0.5	99.3±0.7
12	89.2±1.6	91.1±1.5	88.1±1.6	94.4±1.2	90.0±1.6	90.1±1.3
13	82.5±1.4	90.6±1.2	78.0±1.7	88.4±1.2	87.1±1.9	89.7±1.2
14	73.5±4.3	68.8±4.7	70.8±5.1	63.4±5.8	68.8±4.7	71.3±5.1
15	79.9±5.5	79.2±3.7	79.0±3.5	75.4±6.2	69.0±3.6	78.4±4.6
16	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0
17	57.2±2.5	54.2±3.5	50.6±1.7	50.0±0.0	56.1±4.0	65.1±2.8
18	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.8±0.3	100.0±0.0
19	71.6±3.6	80.0±1.6	70.2±2.4	79.3±1.6	76.2±1.6	79.8±2.1
20	75.0±4.0	76.9±3.0	70.2±4.8	53.7±3.4	65.1±3.0	72.9±3.0
21	67.7±4.9	62.2±3.8	50.6±1.2	61.2±5.8	67.2±5.4	61.3±4.6
22	85.8±2.6	85.0±2.4	82.0±6.4	89.2±2.1	81.8±3.8	91.8±2.3
23	64.6±3.6	57.3±3.2	52.3±2.8	52.6±2.4	61.7±3.4	57.3±3.1
24	96.2±1.3	97.2±0.8	96.0±1.1	97.5±0.7	96.4±1.5	98.1±0.6
25	62.7±3.8	63.5±3.2	60.8±3.4	49.9±0.1	59.4±3.6	58.2±2.6
Mean	79.5±2.6	80.0±2.4	75.2±2.4	78.0±2.4	78.2±2.7	82.2±2.2

Table A.8: Out-of-sample F1 score (%) averaged over 20 train/test splits.

Dataset ID	CoDaCoRe (Defaults)	selbal	Pairwise log-ratios	Coda-lasso	amalgam	Random Forest
1	46.9±2.8	53.8±1.2	10.9±6.5	65.3±1.8	58.6±2.6	65.7±2.8
2	85.6±2.6	68.0±2.1	86.0±3.6	86.4±3.8	87.1±3.6	89.1±2.8
3	84.1±1.8	84.6±0.0	84.6±0.1	73.3±2.9	82.2±1.6	88.4±0.9
4	56.8±3.6	65.7±1.1	65.0±3.1	65.0±3.6	66.4±4.0	71.7±3.0
5	89.6±1.4	84.7±0.1	88.7±0.9	81.0±3.5	87.9±1.7	91.8±1.4
6	60.9±4.5	66.2±1.6	63.6±4.6	66.2±3.5	56.9±4.5	70.4±3.9
7	69.0±3.9	72.9±1.7	10.6±9.5	70.0±7.8	60.3±3.6	69.3±5.2
8	94.3±1.2	80.2±0.9	91.2±1.7	94.6±1.0	93.7±1.2	97.0±0.8
9	68.3±2.5	76.8±0.7	74.2±0.3	62.2±10.2	70.9±2.9	76.6±1.6
10	72.1±1.3	78.6±0.4	76.9±0.0	73.6±4.0	70.6±2.4	77.9±0.7
11	80.8±2.8	38.9±3.9	93.5±1.8	87.0±2.5	94.9±1.3	95.2±1.4
12	96.2±0.3	95.9±0.0	95.9±0.1	91.8±0.7	96.1±0.3	96.0±0.1
13	54.7±2.1	42.7±0.1	31.6±3.9	68.2±1.9	64.5±2.8	62.7±3.5
14	63.2±3.9	64.1±1.1	52.9±9.8	38.1±13.7	60.5±4.3	53.0±6.9
15	77.5±4.4	72.5±1.2	75.9±3.0	75.0±4.7	69.8±3.5	75.9±4.0
16	100.0±0.0	94.1±0.0	100.0±0.0	100.0±0.0	100.0±0.0	99.9±0.1
17	58.4±3.6	69.8±1.2	69.4±1.1	0.0±0.0	57.0±4.2	65.8±2.9
18	100.0±0.0	66.3±0.2	100.0±0.0	100.0±0.0	99.5±0.5	99.8±0.4
19	64.5±2.5	69.0±0.7	65.7±2.1	73.0±1.8	71.3±1.7	73.4±2.3
20	69.5±3.6	71.9±1.2	68.9±2.6	14.8±11.9	60.5±3.6	66.9±3.1
21	62.1±4.5	67.1±2.5	68.2±0.4	34.1±14.8	59.3±5.9	55.8±6.1
22	76.8±3.1	72.7±2.2	74.4±6.4	77.4±2.6	73.2±4.1	88.1±1.9
23	60.8±5.0	64.1±1.0	7.3±4.8	10.2±7.8	51.6±3.6	50.4±3.3
24	95.4±0.9	85.8±0.8	95.2±0.9	95.8±0.8	95.6±0.6	95.7±0.8
25	54.7±3.8	66.8±1.0	43.0±8.0	2.5±5.0	59.7±4.2	55.5±2.9
Mean	73.7±2.6	70.9±1.1	67.8±3.0	64.2±4.4	73.9±2.8	77.3±2.5

A.5 Software Package

We provide open source implementations of CoDaCoRe as R[10] and Python[11] packages. Package documentation and examples, on which this Appendix is based, may be found in the *CoDaCoRe Guide*.[12]

A.5.1 Installation

You can install `codacore` by running:

```
install.packages("codacore")
```

You may instead install the development version directly from Github, using the devtools package.

```
devtools::install_github("egr95/R-codacore", ref="main")
```

Note that CoDaCoRe requires a working installation of TensorFlow. If you do not have Tensorflow previously installed, when you run `codacore()` for the first time you will likely encounter an error message of the form:

```
> codacore(x, y)


ERROR: Could not find a version that satisfies the requirement te
nsorflow
ERROR: No matching distribution found for tensorflow
Error: Installation of TensorFlow not found.


Python environments searched for 'tensorflow' package:
```

[10] https://github.com/egr95/R-codacore
[11] https://github.com/egr95/py-codacore
[12] https://egr95.github.io/R-codacore/guide.html

```
 /moto/stats/users/eg2912/miniconda3/envs/r-test/bin/python3.9
 /usr/bin/python2.7

You can install TensorFlow using the install_tensorflow() functio
n.
```

This can be fixed simply by installing tensorflow, as follows:

```
install.packages("tensorflow")
library("tensorflow")
install_tensorflow()


install.packages("keras")
library("keras")
install_keras()
```

Note also that you may have to restart your R session between installation of `codacore`, `tensorflow`, and `keras`.

A.5.2 Training the Model

We assume a working installation of `codacore` (link).

```
library("codacore")
help(codacore)
```

In this tutorial, we will showcase `codacore` using three datasets that were also analyzed by the authors of `selbal` (Rivera-Pinto et al., 2018). First, we consider the Crohn's disease data from (Gevers et al., 2014).

```
data("Crohn")
x <- Crohn[, -ncol(Crohn)]
y <- Crohn[, ncol(Crohn)]
```

Our goal is to identify ratio-based biomarkers that are predictive of disease status. Our input variable consists of the abundance of 48 microbial species in 975 samples.

```
dim(x)
#> [1] 975  48
```

The output variable is a binary indicator (CD stands for Chron's disease).

```
table(y)
#> y
#>  CD  no
#> 662 313
```

Prior to fitting CoDaCoRe, we must impute any zeros in our input variable (a standard pre-processing step for ratio-based methods).

```
x <- x + 1
```

Next, we split our data into a training and a test set (to keep things simple we do this naively at random, though in practice one might consider stratified sampling and class rebalancing).

```
# For reproducibility, we set a random seed (including in
# TensorFlow, used by codacore)
set.seed(0)
library(tensorflow)
tf$random$set_seed(0)
trainIndex <- sample(1:nrow(x), 0.8 * nrow(x))
```

```
xTrain <- x[trainIndex, ]
yTrain <- y[trainIndex]
```

We are ready to fit CoDaCoRe. We stick to the default parameters for now. Notice the fast runtime (as compared to, for example, `selbal.cv`).

```
model <- codacore(
  xTrain,
  yTrain,
  logRatioType = 'balances', # can also use 'amalgamations'
  lambda = 1                 # regularization parameter (1 corres
  ponds to "1SE rule")
)
```

A.5.3 Visualizing Results

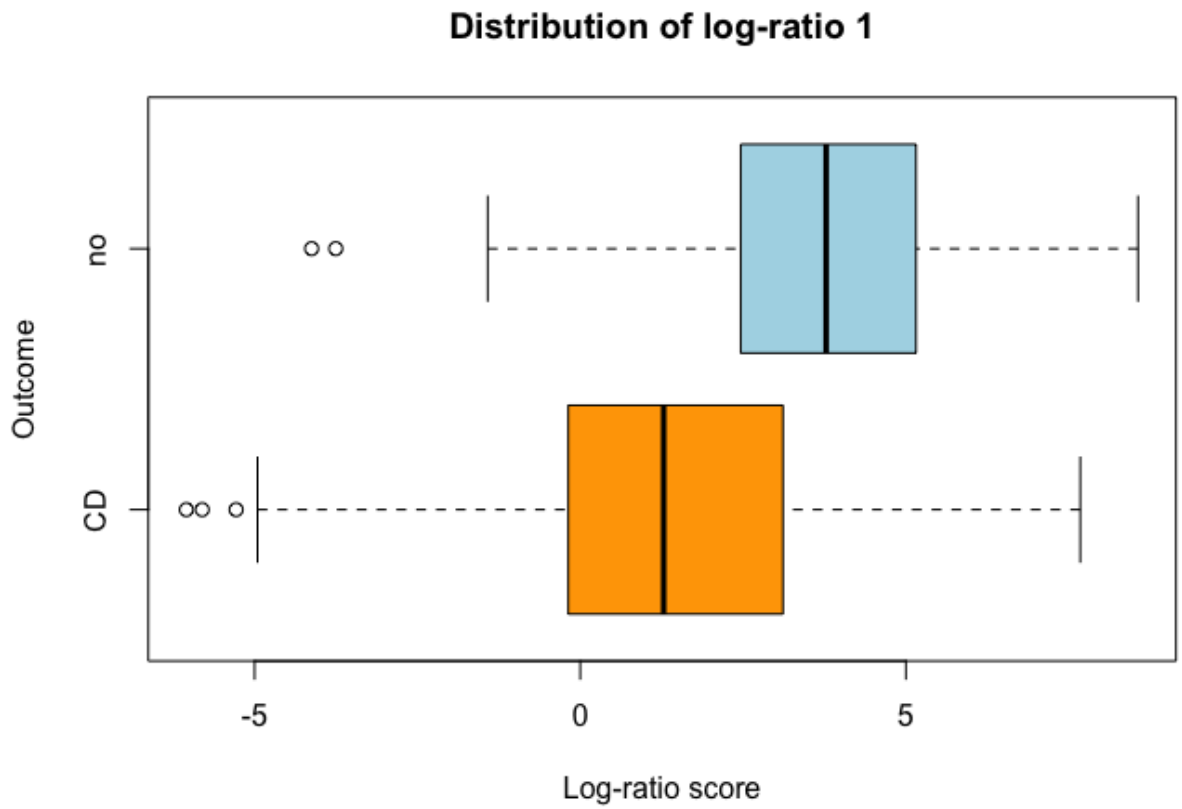
Next we can check the learned output of the model: what inputs were included in the learned log-ratios, how strongly associated they are to the response, and how well they classified the data.

```
print(model)
#>
#> Number of log-ratios found: 2
#> ***
#> Log-ratio rank 1
#> Numerator: g__Roseburia o__Clostridiales_g__
#> Denominator: g__Dialister g__Aggregatibacter
#> AUC: 0.7567907
#> Slope: 0.3939618
#> ***
```

```
#> Log-ratio rank 2
#> Numerator: g__Parabacteroides f__Peptostreptococcaceae_g__ g__
Bacteroides g__Faecalibacterium g__Roseburia g__Lachnospira o__Cl
ostridiales_g__ f__Rikenellaceae_g__
#> Denominator: g__Eggerthella g__Dialister g__Streptococcus g__A
ggregatibacter
#> AUC: 0.778969
#> Slope: 0.3002488
```

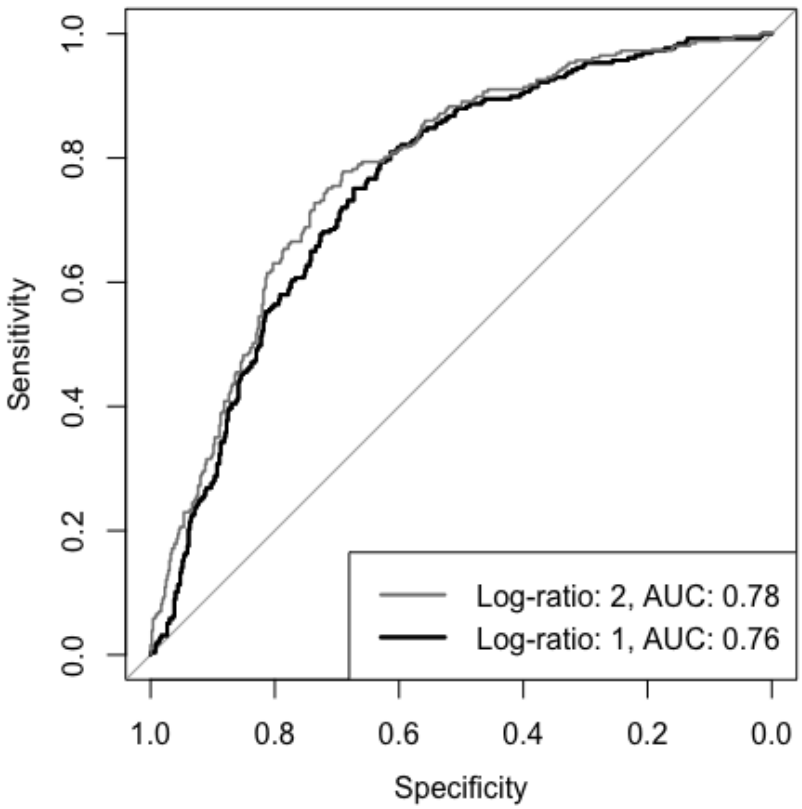
The most predictive ratio identified by CoDaCoRe is Roseburia / Dialister, which can be visualized with the `plot` function.

```
plot(model)
```



Note that CoDaCoRe is an ensemble model, where multiple log-ratios are learned sequentially in decreasing order of importance (with automatic stopping whenever no additional log-ratio improved the loss function during training). We can visualize the performance of this ensembling procedure by "stacking" the respective ROC curves.

```
plotROC(model)
```



A.5.4 Predicting on New Data

We can also use our trained model to classify new samples.

```
xTest <- x[-trainIndex, ]
yTest <- y[-trainIndex]
yHat <- predict(model, xTest, logits = F)
cat("Test set AUC =", pROC::auc(pROC::roc(yTest, yHat,
```

```
    quiet = T)))
#> Test set AUC = 0.80537
# Convert probabilities into a binary class
failure <- yHat < 0.5
success <- yHat >= 0.5
yHat[failure] <- levels(y)[1]
yHat[success] <- levels(y)[2]
cat("Classification accuracy on test set =", round(mean(yHat ==
    yTest), 2))
#> Classification accuracy on test set = 0.73
```

Note our `predict` function can be restricted to only use the top k log-ratios in the model for prediction. For example, the following will compute the AUC of a 1-log-ratio model, using only the top log-ratio.

```
yHat <- predict(model, xTest, logits = F, numLogRatios = 1)
cat("Test set AUC =", pROC::auc(pROC::roc(yTest, yHat,
    quiet = T)))
#> Test set AUC = 0.7859712
```

Other useful functions include:

```
getNumeratorParts(model, 1)
getDenominatorParts(model, 1)
getLogRatios(model, xTest)
getNumLogRatios(model)
getTidyTable(model)
getSlopes(model)
```

A.5.5 Controlling Overlap

By default, CoDaCoRe allows for "overlapping log-ratios", in other words, an input variable that is included in the first log-ratio may well be included in a second or third log-ratio provided it is sufficiently predictive. However, the user may choose to restrict each successive log-ratio to be constructed from a mutually exclusive set of input variables (e.g., to obtain *orthogonal balances*, in the Aitchison sense). This can be specified with the parameter `overlap`. In our example, note how `g__Dialister` is no longer repeated.

```
model <- codacore(xTrain, yTrain, overlap = F)
print(model)
#>
#> Number of log-ratios found: 2
#> ***
#> Log-ratio rank 1
#> Numerator: g__Roseburia o__Clostridiales_g__
#> Denominator: g__Dialister g__Aggregatibacter
#> AUC: 0.7567907
#> Slope: 0.3939618
#> ***
#> Log-ratio rank 2
#> Numerator: g__Parabacteroides f__Peptostreptococcaceae_g__ g__
Bacteroides g__Faecalibacterium g__Lachnospira f__Rikenellaceae_g
__
#> Denominator: g__Eggerthella f__Enterobacteriaceae_g__
#> AUC: 0.7712166
#> Slope: 0.1858207
```

A.5.6 Using Amalgamations

CoDaCoRe can be used to learn log-ratios between both geometric means (known as "balances" or "isometric-log-ratio") or summations (known as "amalgamations" or "summed-log-ratio"), depending on the goals of the user. This can be specified with the parameter `logRatioType`.

```
model <- codacore(xTrain, yTrain, logRatioType = "amalgamations")
print(model)
#>
#> Number of log-ratios found: 1
#> ***
#> Log-ratio rank 1
#> Numerator: g__Parabacteroides g__Bacteroides g__Faecalibacteri
um o__Clostridiales_g__
#> Denominator: g__Haemophilus g__Blautia f__Enterobacteriaceae_g
__ g__Dialister g__Streptococcus g__Fusobacterium
#> AUC: 0.7104552
#> Slope: 0.4062366
```

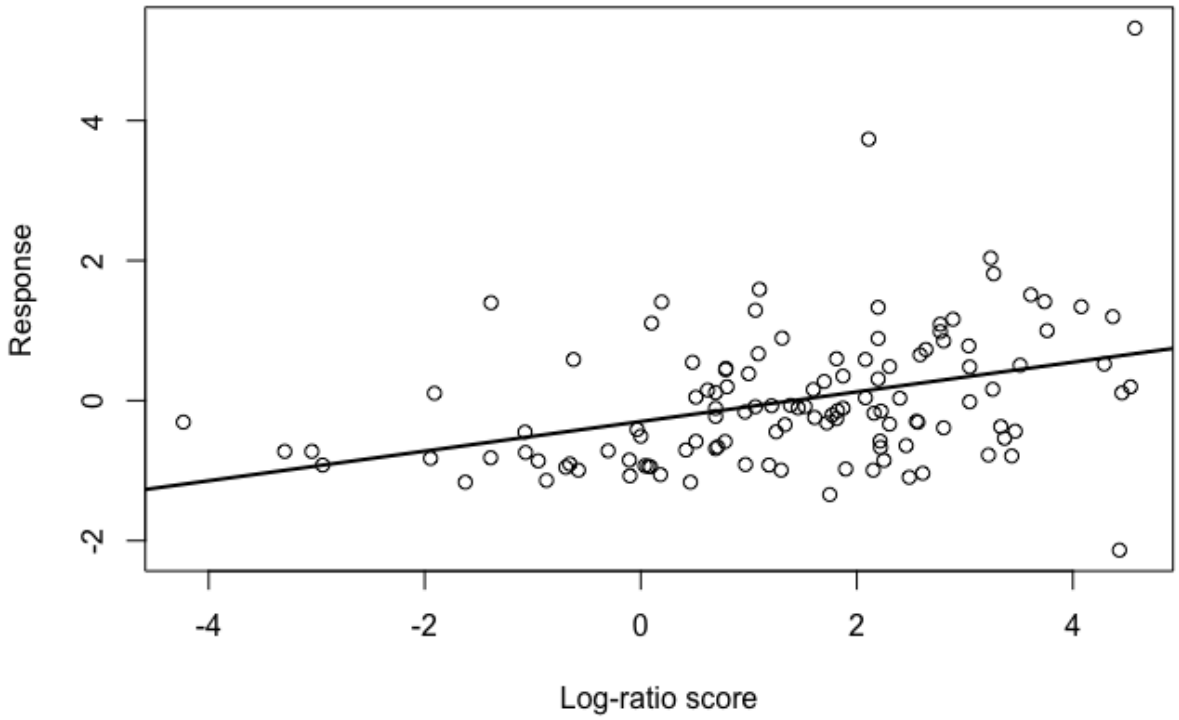
Note that amalgamations/summed-log-ratios are less sensitive to covariates that are small in magnitude (e.g., rare microbes), which can hinder their predictive strength for datasets where small covariates are important. On the other hand, summed-log-ratios have a different interpretation than isometric-log-ratios and may therefore be preferrable in some applications (e.g., when the "summed" effect of an aggregated sub-population is the object of interest). In our Crohn's disease data, the rare species Roseburia gets picked up by the isometric-log-ratio, but not by the summed-log-ratio, which is more sensitive to more common bacteria species such as Faecalibacterium.

A.5.7 Continuous Outcomes

We consider the HIV data from (Noguera-Julian et al., 2016). The goal here is to construct a log-ratio of the microbial abundances that is predictive of the inflammation marker "sCD14", a continuous response variable. CoDaCoRe can be applied much in the same way, except the loss function changes from binary cross-entropy to mean-squared-error. This change will happen automatically based on the values inputted as `y` (although it can also be overriden manually via the `objective` parameter, for example, if the user wanted to fit a binary response using the mean-squared-error, they could specify `objective = 'regression'`).

```
data("sCD14")
x <- sCD14[, -ncol(sCD14)]
y <- sCD14[, ncol(sCD14)]

# Replace zeros as before
x <- x + 1

# Split the data
trainIndex <- sample(1:nrow(x), 0.8 * nrow(x))
xTrain <- x[trainIndex, ]
yTrain <- y[trainIndex]

# Fit codacore and inspect results
model <- codacore(xTrain, yTrain)
print(model)
#>
#> Number of log-ratios found: 1
#> ***
```

```
#> Log-ratio rank 1
#> Numerator: g_Subdoligranulum
#> Denominator: g_Bifidobacterium
#> R squared: 0.1438051
#> Slope: 611.0956
plot(model)
```



A.5.8 Tuning the Regularization

The parameter `lambda` controls the regularization strength of CoDaCoRe. In particular, `lambda = 1` (the default value) corresponds to applying the 1-standard-error rule in the discretization step of the log-ratio (details in Section 3.3). This is typically a good choice, leading to models that are both sparse and predictive. Sparser models can be achieved by higher values of `lambda`, for example, `lambda = 2` corresponds to applying a "2-standard-error" rule. On

the other hand, smaller values of lambda result in less sparse, but typically most predictive, models. In particular, `lambda = 0` corresponds to a "0 standard-error rule", in other words choosing the log-ratio that minimizes cross-validation score. Such a choice can be good when we seek a maximally predictive model, but care less about sparsity.

```
model <- codacore(xTrain, yTrain, lambda = 0)
print(model)
#>
#> Number of log-ratios found: 3
#> ***
#> Log-ratio rank 1
#> Numerator: g_Subdoligranulum g_Dialister g_Paraprevotella f_De
fluviitaleaceae_g_Incertae_Sedis
#> Denominator: f_Lachnospiraceae_g_unclassified g_Succinivibrio
g_Parabacteroides g_Lachnospira g_Coprococcus g_Catenibacterium g
_Roseburia g_Megasphaera g_Mitsuokella g_Sutterella g_Bifidobacte
rium g_Streptococcus f_Erysipelotrichaceae_g_unclassified f_Porph
yromonadaceae_g_unclassified g_Collinsella
#> R squared: 0.319985
#> Slope: 2040.222
#> ***
#> Log-ratio rank 2
#> Numerator: g_Faecalibacterium g_Bacteroides g_Alloprevotella g
_Subdoligranulum g_Clostridium_sensu_stricto_1 g_Escherichia-Shig
ella f_Defluviitaleaceae_g_Incertae_Sedis
#> Denominator: f_Lachnospiraceae_g_unclassified g_Lachnospira g_
Barnesiella g_Catenibacterium g_Megasphaera g_Mitsuokella g_Sutte
rella g_Bifidobacterium g_Collinsella
```

```
#> R squared: 0.3852063
#> Slope: 816.7075
#> ***
#> Log-ratio rank 3
#> Numerator: g_Alloprevotella
#> Denominator: g_Bifidobacterium
#> R squared: 0.390754
#> Slope: 66.46405
```

Notice the increased R-squared score relative to the previous model (at the expense of sparsity).

On some datasets, CoDaCoRe may have trouble finding *any* predictive log-ratios. If none are found, this is typically a sign that the signal in the data is weak. In this case, the analyst may choose to reduce the value of `lambda` (for example, to `lambda = 0`), in order to allow our algorithm to search more aggressively for predictive log-ratios. Doing so will often allow the algorithm to identify at least one predictive log-ratio, at the risk of overfitting the training data. Additional care must be taken in validating such log-ratios on held-out data.

A.5.9 Covariate Adjustment

Many applications require accounting for potential confounder variables as well as our ratio-based biomarkers. As an example, we consider a second HIV dataset from (Noguera-Julian et al. 2016). The goal is to find a microbial signature for HIV status, i.e., a log-ratio that can discriminate between HIV-positive and HIV-negative individuals. However, we have an additional confounder variable, MSM (Men who have Sex with Men). In the context of CoDaCoRe, there are multiple approaches that can be used to adjust for covariates.

A.5.10 Incremental Fit

Given the *stagewise-additive* (i.e., ensemble) nature of CoDaCoRe, whereby each successive log-ratio is fitted on the residual of the previous iteration, a very natural approach is to fit the

covariates *a priori* and then fit CoDaCoRe on the residual. In other words, we would start by regressing HIV status on MSM, "partialling out" this covariate, and then fit CoDaCoRe on the residual from this model. This can be implemented easily by means of the `offset` parameter.

```
data("HIV")
x <- HIV[, 1:(ncol(HIV) - 2)]
z <- HIV[, "MSM"]
y <- HIV$HIV_Status

# Replace zeros as before
x <- x + 1

# Split the data
trainIndex <- sample(1:nrow(x), 0.8 * nrow(x))
dfTrain <- HIV[trainIndex, ]
xTrain <- x[trainIndex, ]
yTrain <- y[trainIndex]

partial <- glm(HIV_Status ~ MSM, data = dfTrain,
               family = "binomial")
# Note the offset must be given in logit space
model <- codacore(xTrain, yTrain, offset = predict(partial))
print(model)
#>
#> Number of log-ratios found: 1
#> ***
#> Log-ratio rank 1
#> Numerator: g_Bacteroides g_Anaerovibrio
```

```
#> Denominator: g_Prevotella g_Alloprevotella g_RC9_gut_group g_C
atenibacterium g_Dialister f_Ruminococcaceae_g_Incertae_Sedis f_v
adinBB60_g_unclassified g_Oribacterium
#> AUC: 0.8019231
#> Slope: 0.5233792
partialAUC <- pROC::auc(pROC::roc(yTrain, predict(partial),
    quiet = T))
codacoreAUC <- model$ensemble[[1]]$AUC
cat("AUC gain:", round(100 * (codacoreAUC - partialAUC)),
    "%")
#> AUC gain: 16 %
```

Note that, when predicting on new data, the contributions of the covariates and the log-ratios should be added up in logit space.

```
dfTest <- HIV[-trainIndex, ]
xTest <- x[-trainIndex, ]
yTest <- z[-trainIndex]
yHatLogit <- predict(partial, newdata = dfTest) + predict(model,
    xTest, logits = T)
yHat <- yHatLogit > 0  # in case we need binary predictions e.g.
to compute accuracy
testAUC <- pROC::auc(pROC::roc(yTest, yHatLogit, quiet = T))
cat("Test AUC:", round(100 * testAUC), "%")
#> Test AUC: 100 %
```

When the outcome variable is continuous, this is simpler as there is no logit transformation and the contributions of the partial model can be added directly, e.g.,

```
# Suppose that, instead of predicting HIV status (a binary
#  target), we now have some continuous target, 'yCts'
partial2 <- lm(yCts ~ MSM, data=dfTrain)
model2 <- codacore(xTrain, yCtsTrain, offset=predict(partial))
print(model2)
yCtsHat <- predict(partial2, newdata = dfTest) + predict(model2,
    xTest)
MSE <- mean((yCtsTest - yCtsHat)^2)
```

A.5.11 Joint Fit

Depending on the application and the goals of the analyst, it may be of interest to understand the *joint* effect of the covariates and log-ratios on the response. To do so, one option is to simply regress the outcome jointly against the covariates and the learned log-ratios from the previous step. This can be implemented by running, in addition to the above, an additional `glm` fit.

```
# Create a new design matrix with response & covariates, as well
# as log-ratios obtained from codacore
dfJoint = cbind(dfTrain[, c("MSM", "HIV_Status")],
    getLogRatios(model))

# And fit everything jointly
modelJoint <- glm(HIV_Status ~ ., data = dfJoint,
    family = "binomial")
# Can again use this model to make predictions or to interpret
# regression coefficients
yHat <- predict(modelJoint, newData = dfJoint)
summary(modelJoint)
```

```
#>
#> Call:
#> glm(formula = HIV_Status ~ ., family = "binomial", data = dfJo
int)
#>
#> Deviance Residuals:
#>     Min       1Q   Median       3Q      Max
#> -2.6502   0.1875   0.3587   0.6123   1.1661
#>
#> Coefficients:
#>              Estimate Std. Error z value Pr(>|z|)
#> (Intercept)    1.1801     0.7053   1.673 0.094268 .
#> MSMMSM         0.6863     0.8905   0.771 0.440916
#> `log-ratio1`   0.7671     0.2226   3.446 0.000568 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 109.567  on 123  degrees of freedom
#> Residual deviance:  88.559  on 121  degrees of freedom
#> AIC: 94.559
#>
#> Number of Fisher Scoring iterations: 6
```

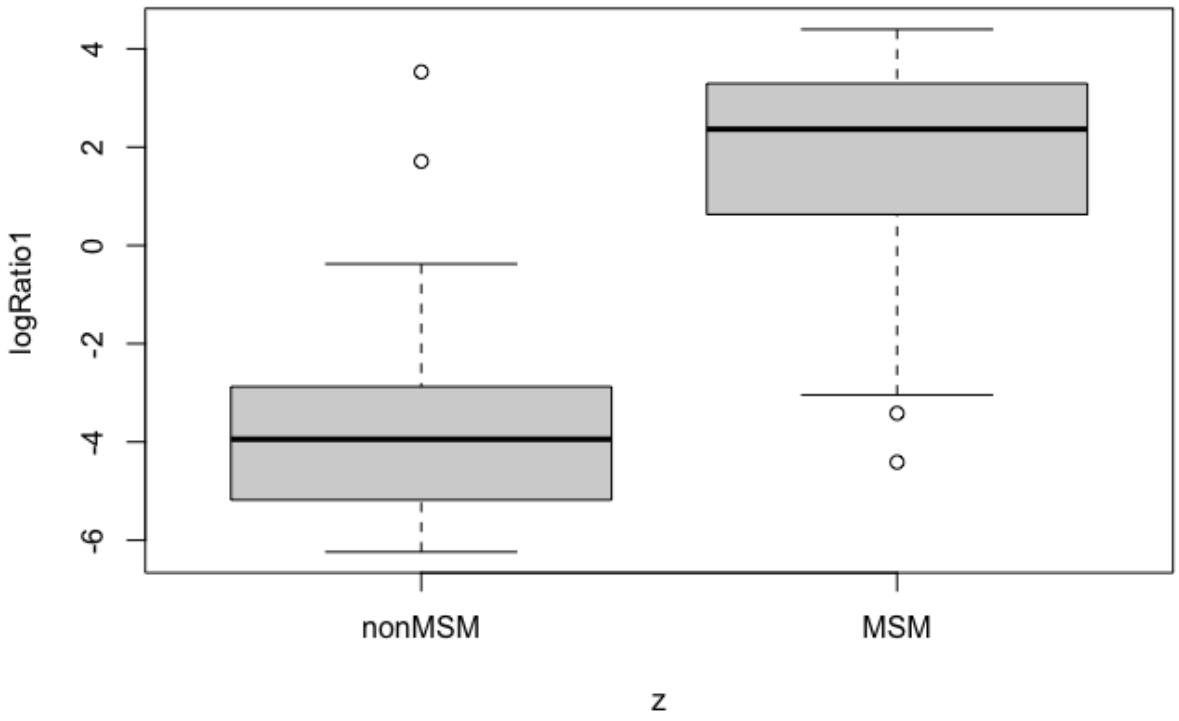
Note that, in any case, the CoDaCoRe algorithm itself only optimizes over one log-ratio at a time (in its current implementation). In some applications, it may in fact be beneficial to optimize over the set of log-ratios jointly with the regression coefficients of the covariates. However, this is

not yet implemented.

A.5.12 Unsupervised Learning

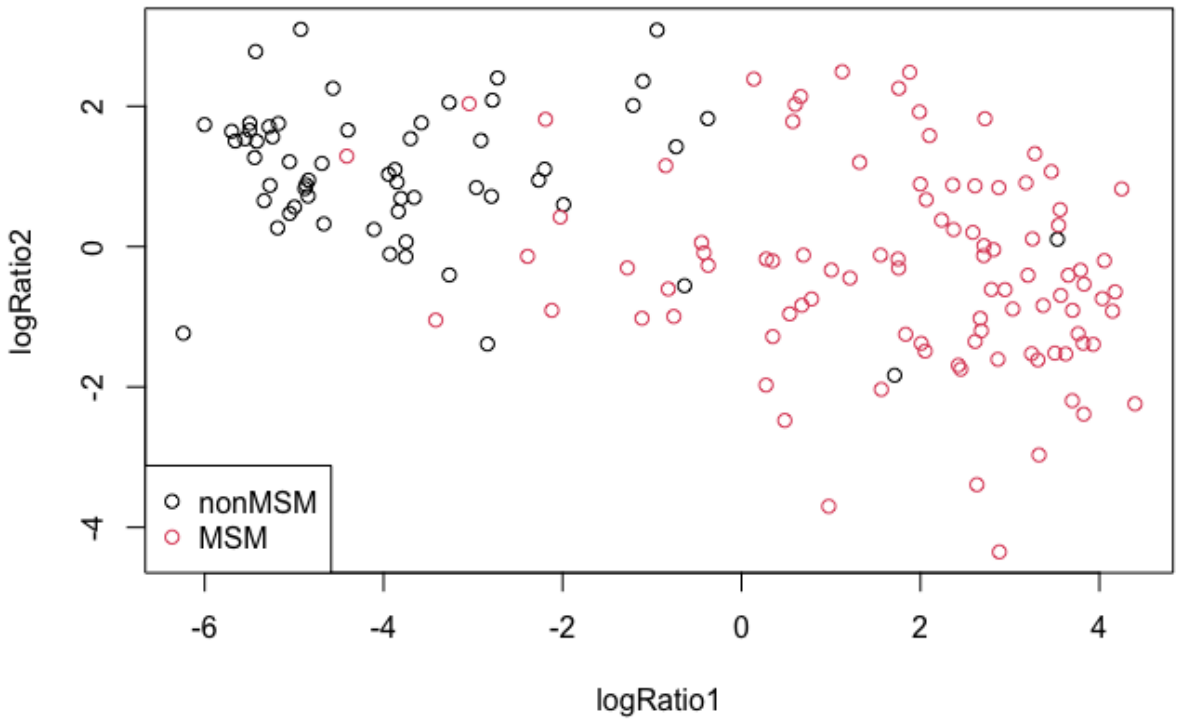
CoDaCoRe can be used as follows to obtain a fast, scalable, interpretable and sparse log-ratio based unsupervised learning algorithm. The idea is to first compute a dense representation of the data using traditional methods, and then regress the data against this representation using CoDaCoRe to obtain a sparse log-ratio representation in its stead. For example, one could take the first principal component of the CLR-transformed data, and use CoDaCoRe to approximate this real-valued representation with a single sparse log-ratio score (Quinn et al., 2021). In the present HIV dataset, we find that the learned log-ratio biomarker provides a useful representation of the data, markedly separating the MSM from the non-MSM individuals.

```
clr <- t(apply(x, 1, function(x) log(x) - mean(log(x))))
pca <- prcomp(clr, scale = T)
pc1 = clr %*% pca$rotation[, 1]

model <- codacore(x, as.numeric(pc1))
logRatio1 <- getLogRatios(model, x)[, 1]
boxplot(logRatio1 ~ z)
```



We can take things one step further and derive a second unsupervised log-ratio biomarker, by simply fitting CoDaCoRe on the second principal component. Taken together, our two log-ratio biomarkers capture important information in the data:

```
pc2 = clr %*% pca$rotation[, 2]
model <- codacore(x, as.numeric(pc2))
logRatio2 <- getLogRatios(model, x)[, 1]
plot(logRatio1, logRatio2, col = z)
legend("bottomleft", legend = levels(z), pch = 1, col = 1:2)
```



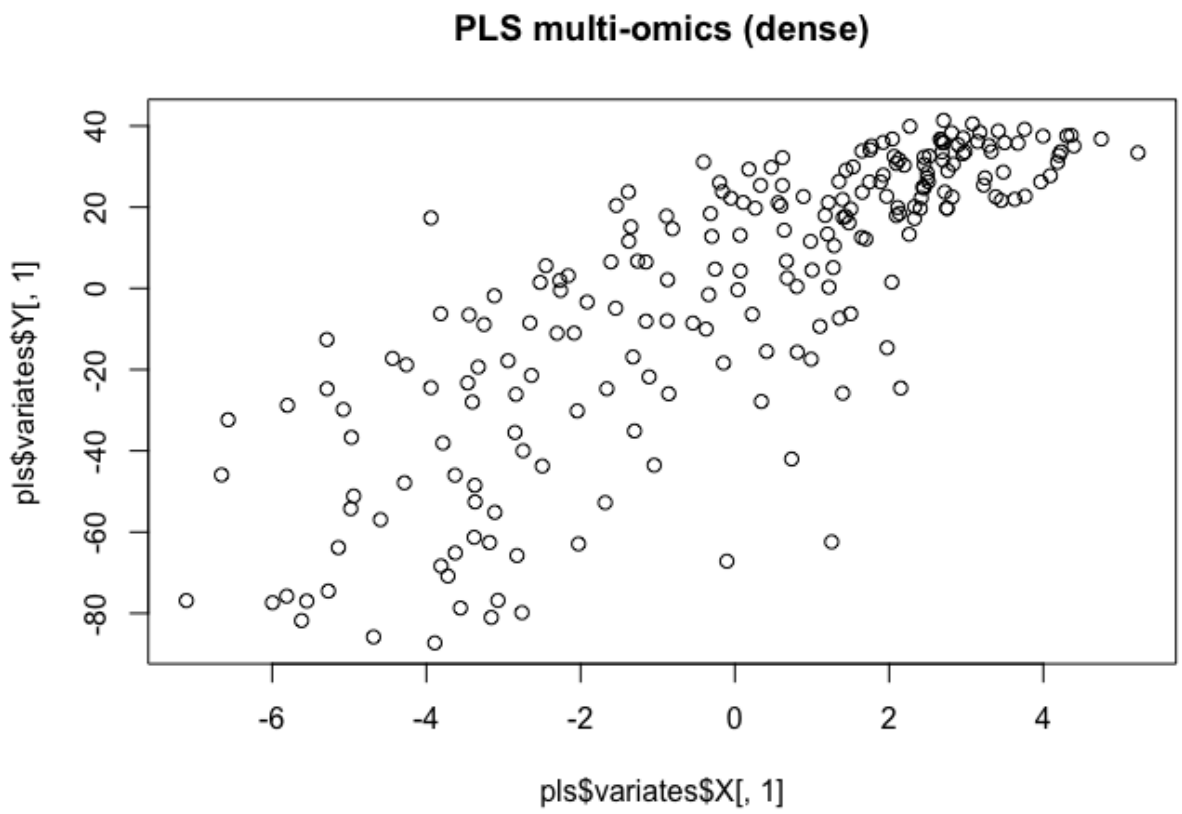
Note also that the CoDaCoRe framework can be applied to the unsupervised learning problem in several other ways, some of which are under active development.

A.5.13 Multi-Omics Integration

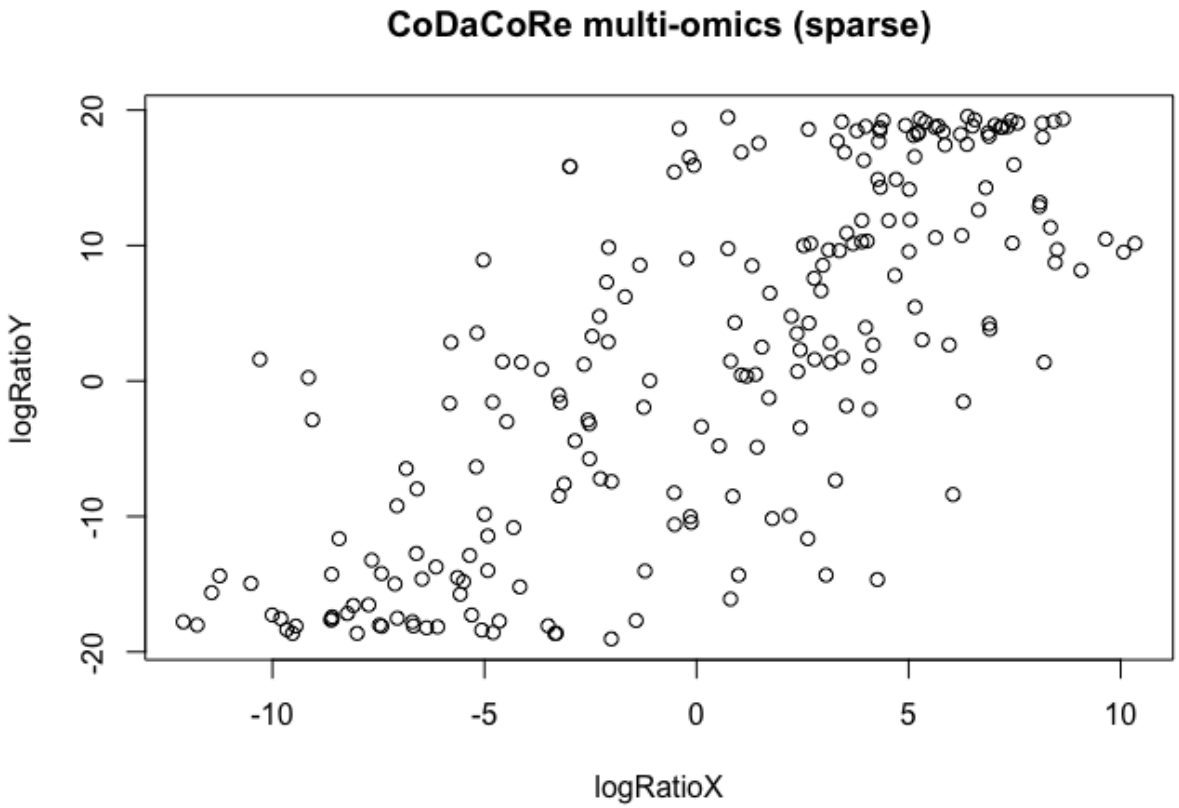
With a similar approach, CoDaCoRe can be used for scalable, sparse, and interpretable multi-omics data integration. We briefly highlight an example multi-omics analysis of paired gut microbiome and metabolomics data, taken from 220 clinical samples of which 88 have Chron's disease and 76 have ulcerative colitis (Franzosa et al., 2019). For a full analysis, see Section 5 and the appendix in Quinn et al., 2021. Again, we will use standard techniques to compute a (dense) latent representation of the data, which we will then approximate using sparse log-ratio biomarkers. Letting $\mathbf{T}$ denote the microbe abundances $\mathbf{U}$ the metabolite abundances, we will use partial least squares (PLS) regression to model the association between $\mathbf{T}$ and $\mathbf{U}$. This will result in two latent factors, one for $\mathbf{T}$ and one for $\mathbf{U}$, that capture the *joint* information in the data. These latent factors

will constitute the regression target for CoDaCoRe.

```
# Load data
download.file("https://github.com/egr95/FranzosaData/blob/main/Fr
anzosaMicrobiome.rda?raw=true", "FranzosaMicrobiome")
download.file("https://github.com/egr95/FranzosaData/blob/main/Fr
anzosaMetabolite.rda?raw=true", "FranzosaMetabolite")
load("FranzosaMicrobiome")
load("FranzosaMetabolite")

# Note data have already been pre-processed as per (Quinn et
# al., 2021), including zero-replacement and normalization to a
# unit total.
T <- FranzosaMicrobiome[, -ncol(FranzosaMicrobiome)]  # We remove
 the last column (response variable)
U <- FranzosaMetabolite[, -ncol(FranzosaMetabolite)]

# Apply clr transform prior to PLS
clrT <- t(apply(T, 1, function(x) log(x) - mean(log(x))))
clrU <- t(apply(U, 1, function(x) log(x) - mean(log(x))))

# Call mixOmics package and plot first PLS components
suppressMessages(library("mixOmics"))
pls <- mixOmics::pls(X = clrT, Y = clrU, ncomp = 1)
plot(pls$variates$X[, 1], pls$variates$Y[,
    1], main = "PLS multi-omics (dense)")
```




```
# Approximate the dense PLS representations with sparse
# log-ratio biomarkers
plsX <- pls$variates$X[, 1]
modelX <- codacore(T, plsX)
logRatioX <- getLogRatios(modelX)[, 1]

plsY <- pls$variates$Y[, 1]
modelY <- codacore(U, plsY, logRatioType = "B")
logRatioY <- getLogRatios(modelY)[, 1]

plot(logRatioX, logRatioY, main = "CoDaCoRe multiomics (sparse)")
```



Note that CoDaCoRe obtains a sparse representation that also has better statistical properties than the original (dense) PLS components, markedly de-skewing the data.

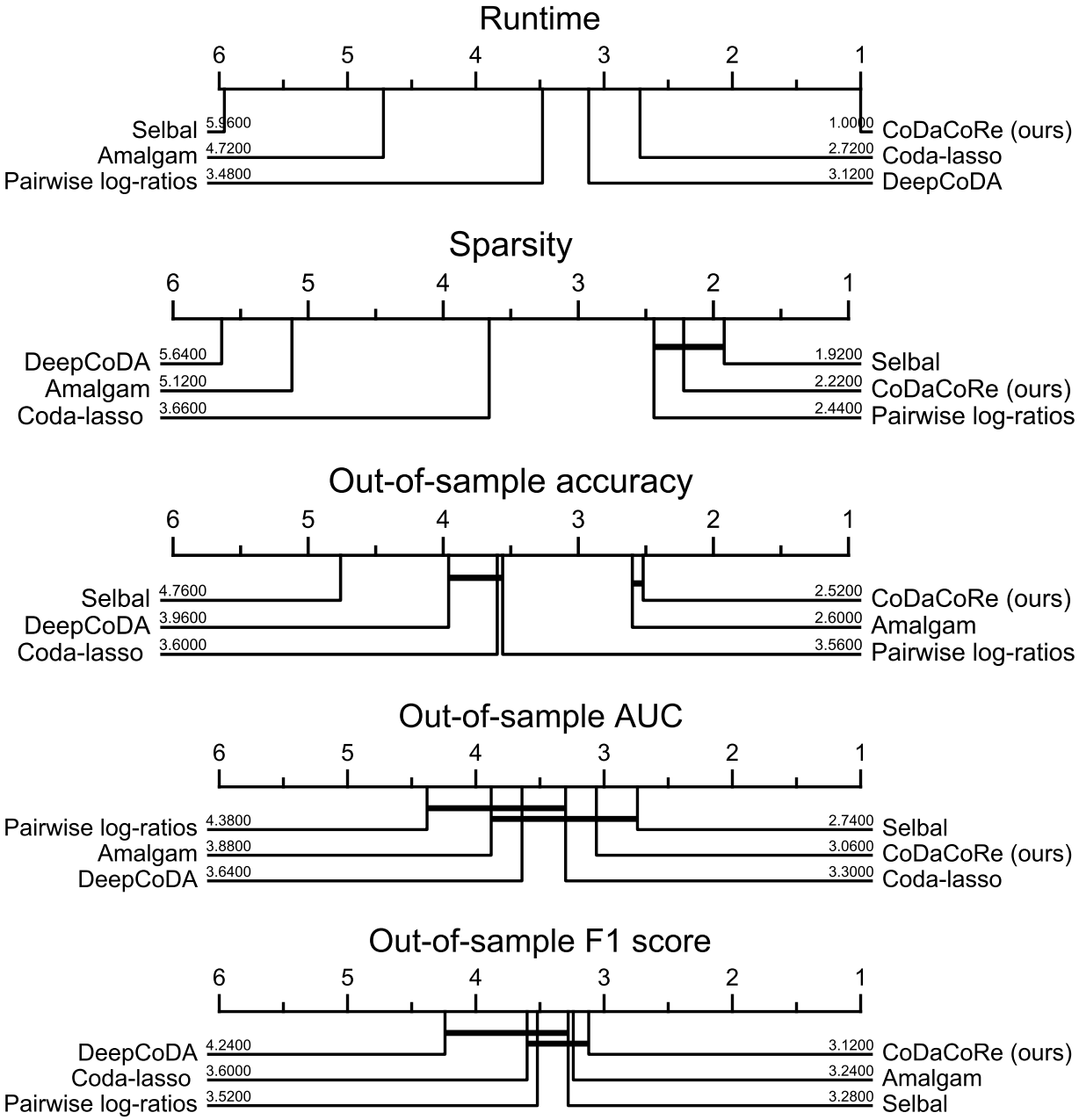


Figure A.2: Critical difference diagrams for 5 different performance metrics, computed using the Wilcoxon-Holm method at 5% significance. CoDaCoRe is the most computationally efficient and the sparsest (tied with selbal and pairwise log-ratios), as well as being tied to the leading position in all 3 metrics of predictive accuracy.

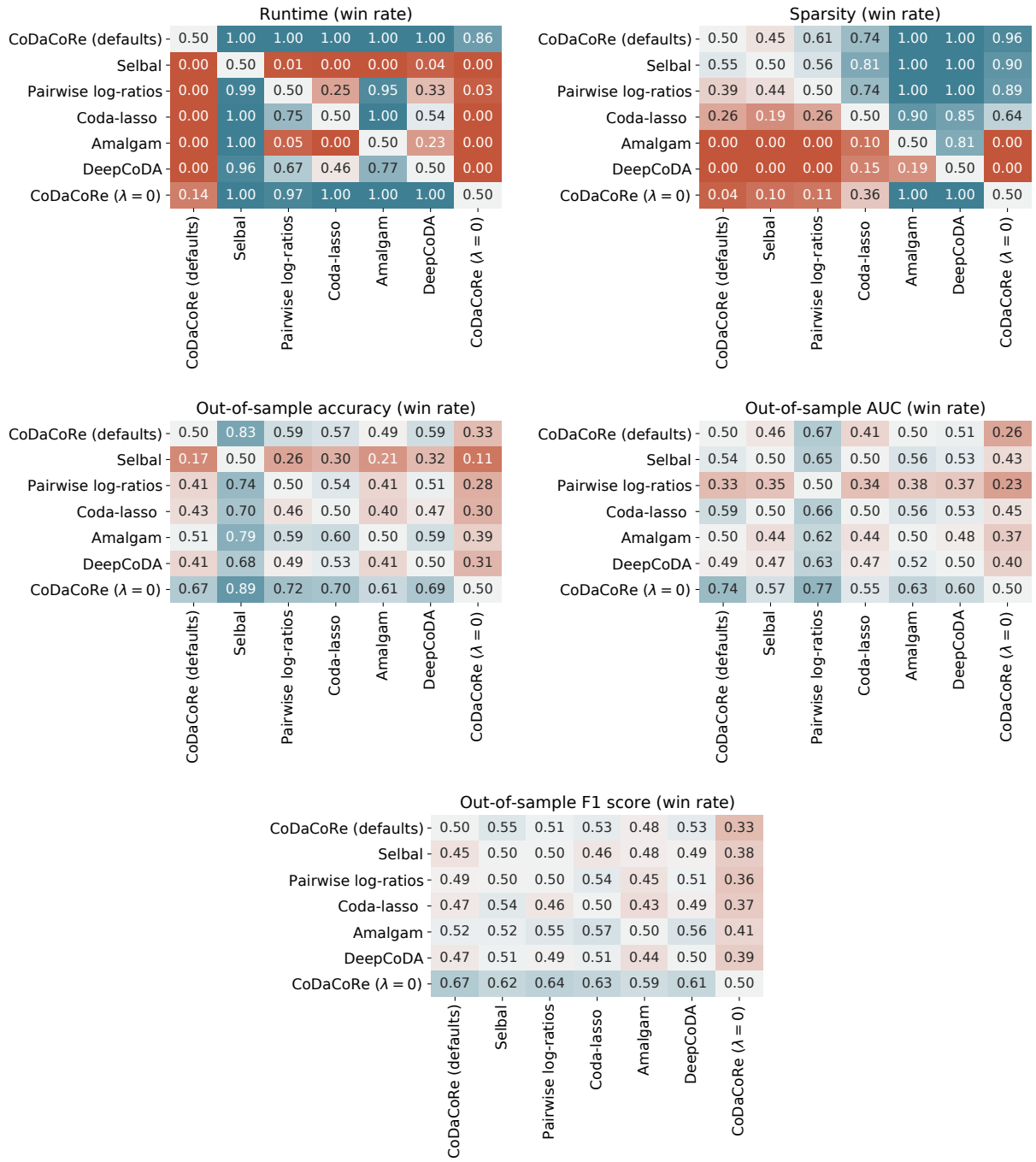


Figure A.3: Pairwise win rates for each evaluation metric. The (i, j)th entry shows the proportion of runs in which the ith method achieved a better score than the jth method, where i indexes rows and j indexes columns. Note that, while CoDaCoRe (with default parameters) substantially outperforms in terms of runtime and sparsity, the only method that consistently outperforms in terms of predictive accuracy is CoDaCoRe with $\lambda = 0$.

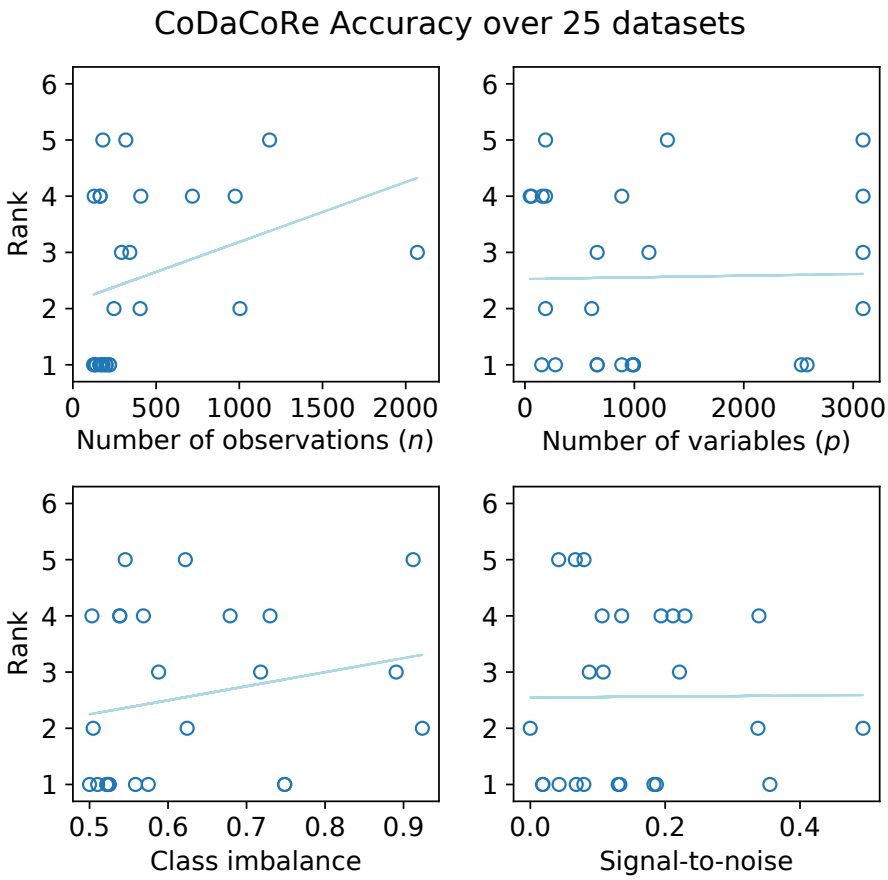


Figure A.4: Rank scores of CoDaCoRe (with default parameters) relative to its competitors regressed against 4 dataset-level metrics of interest. Each point represents a different dataset, of which there are 25 total. Trendlines were obtained from 4 separate univariate least squares fits. Class imbalance is shown as a number between 0.5 and 1, denoting the proportion of datapoints in the majority class. Signal-to-noise ratio is estimated using the out-of-sample accuracy of the Random Forest model (note this is a rough proxy intended for illustrative purposes only). Note that CoDaCoRe tends to outperform the most on datasets with a low number of samples, and balanced classes.

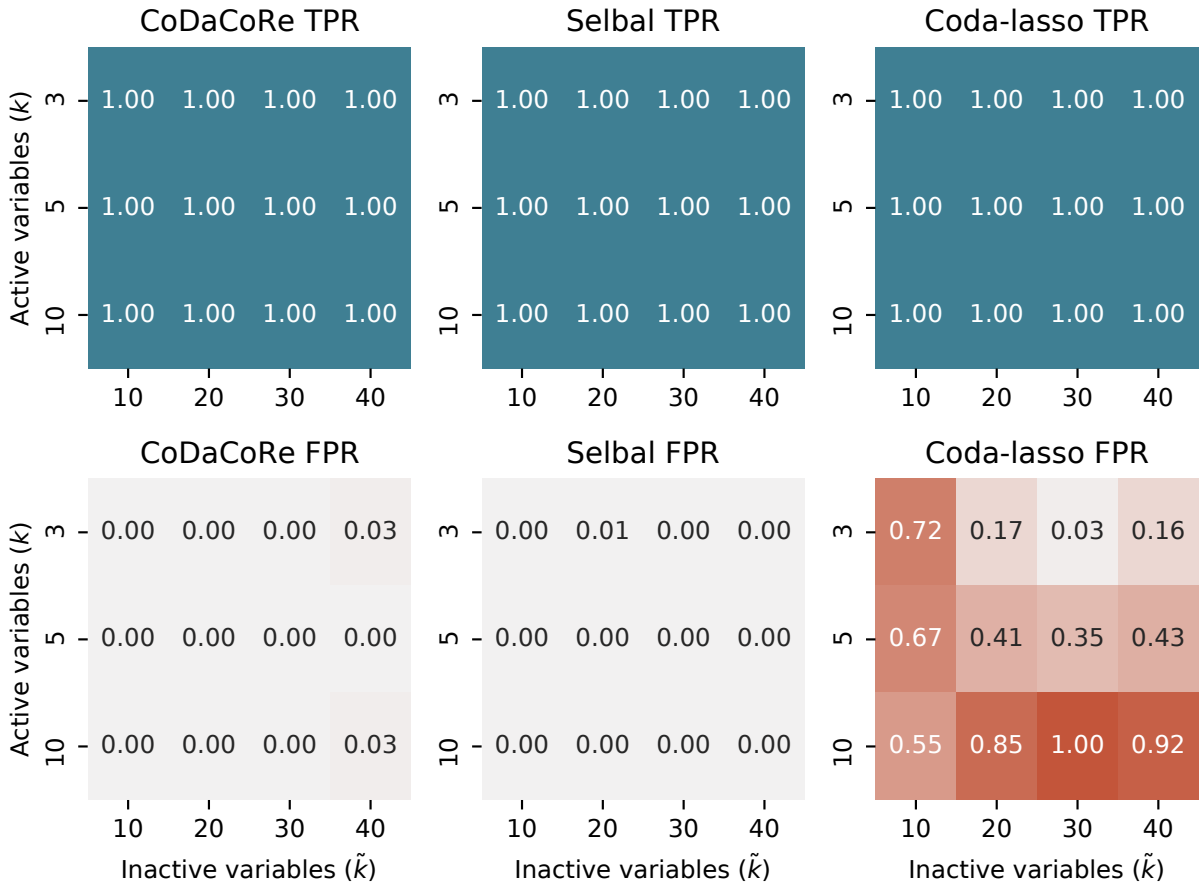


Figure A.5: Variable selection performance for CoDaCoRe, selbal, and coda-lasso for a range of simulated datasets. True positive rates represent the proportion of active variables that were selected by the model, averaged over 10 independent draws of the synthetic dataset for each combination of k (active variables) and $\tilde{k}$ (inactive variables). False positive rates represent the proportion of inactive variables that were selected by the model, averaged over the same draws. In all, $3 \times 4 \times 10 = 120$ datasets were used to produce this table, and the response variables were simulated using balances.

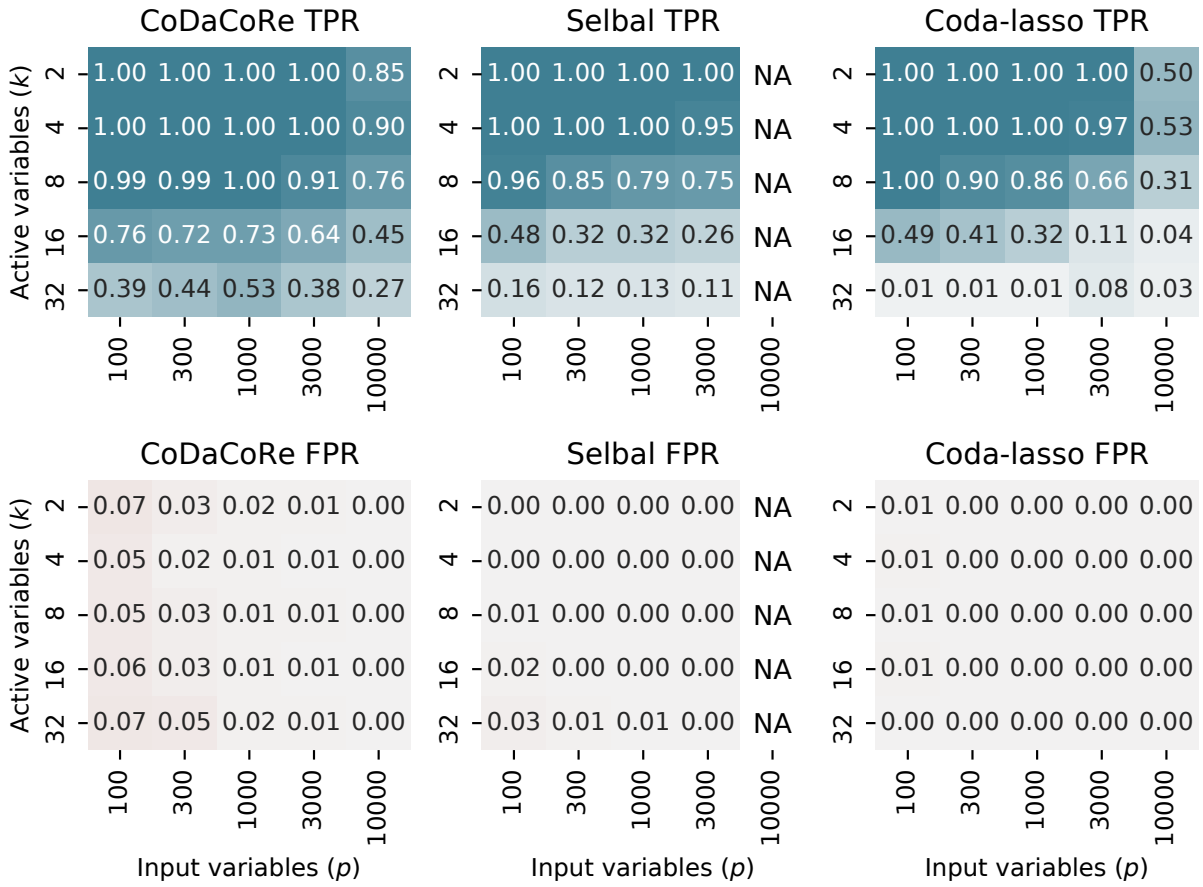


Figure A.6: Variable selection performance for CoDaCoRe, selbal, and coda-lasso for a range of simulated datasets. True positive rates represent the proportion of active variables that were selected by the model, averaged over 10 independent draws of the synthetic dataset for each combination of k (active variables) and p (input variables). False positive rates represent the proportion of inactive variables that were selected by the model, averaged over the same draws. In all, $5 \times 5 \times 10 = 250$ datasets were used to produce this table, and the response variables were simulated using balances.

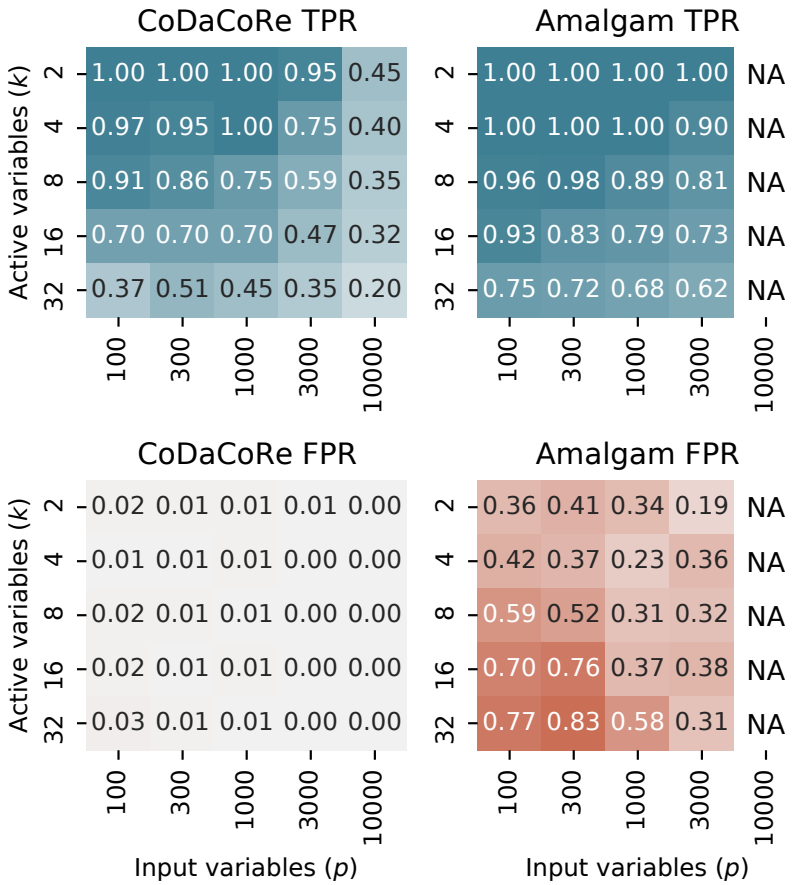


Figure A.7: Variable selection performance for CoDaCoRe and amalgam, for a range of simulated datasets. True positive rates represent the proportion of active variables that were selected by the model, averaged over 10 independent draws of the synthetic dataset for each combination of k (active variables) and p (input variables). False positive rates represent the proportion of inactive variables that were selected by the model, averaged over the same draws. In all, $5 \times 5 \times 10 = 250$ datasets were used to produce this table, and the response variables were simulated using summed log-ratios.

Appendix B: The Continuous Categorical

B.1 Derivation of the Normalizing Constant

For clarity, we start by recalling the expression that we aim to show (equation 5.4 from the main text), and we make the dependence on K explicit by writing $C_K(\boldsymbol{\eta})$:

$$C_K(\boldsymbol{\eta}) = \left((-1)^{K+1} \sum_{k=1}^{K} \frac{\exp(\eta_k)}{\prod_{i \neq k} (\eta_i - \eta_k)} \right)^{-1}. \tag{B.1}$$

Proof: we proceed by induction on K. The base case $K = 2$ can be integrated directly:

$$\begin{aligned} C_2(\boldsymbol{\eta}) &= \left(\int_0^1 \exp(\eta_1 x_1) dx_1 \right)^{-1} \\ &= \left(\frac{e^{\eta_1} - 1}{\eta_1} \right)^{-1} \\ &= \left(-\frac{e^{\eta_1}}{\eta_2 - \eta_1} - \frac{e^{\eta_2}}{\eta_1 - \eta_2} \right)^{-1}, \end{aligned} \tag{B.2}$$

where the last equality follows from $\eta_2 = 0$.

For the inductive step, we assume that equation B.1 gives the correct normalizing constant for $K - 1$, and compute the integral for K:

$$\begin{aligned} C_K(\boldsymbol{\eta})^{-1} &= \int_{\mathbb{S}^{K-1}} \exp(\boldsymbol{\eta}^\top \mathbf{x}) d\mu \\ &= \int_0^1 \int_0^{1-x_1} \cdots \int_0^{1-x_1-\cdots-x_{K-2}} \exp\left(\sum_{i=1}^{K-1} \eta_i x_i \right) dx_{K-1} \cdots dx_2 dx_1. \end{aligned} \tag{B.3}$$

For the innermost integral, we have:

$$\begin{aligned}
&\int_0^{1-x_1-\cdots-x_{K-2}} \exp\left(\sum_{i=1}^{K-1}\eta_i x_i\right) dx_{K-1} \\
&\quad= \exp\left(\sum_{i=1}^{K-2}\eta_i x_i\right)\int_0^{1-x_1-\cdots-x_{K-2}} \exp(\eta_{K-1}x_{K-1})dx_{K-1} \\
&\quad= \exp\left(\sum_{i=1}^{K-2}\eta_i x_i\right)\left[\frac{1}{\eta_{K-1}}\exp(\eta_{K-1}t)\right]_{t=0}^{t=1-x_1-\cdots-x_{K-2}} \\
&\quad= \frac{1}{\eta_{K-1}}\exp\left(\sum_{i=1}^{K-2}\eta_i x_i\right)[\exp(\eta_{K-1}(1-x_1-\cdots-x_{K-2}))-1] \\
&\quad= \frac{1}{(\eta_{K-1}-\eta_K)}\left[\exp(\eta_{K-1})\exp\left(\sum_{i=1}^{K-2}(\eta_i-\eta_{K-1})x_i\right)-\exp\left(\sum_{i=1}^{K-2}\eta_i x_i\right)\right]. \qquad \text{(B.4)}
\end{aligned}$$

Letting $\eta_i^{(1)} = \eta_i - \eta_{K-1}$ for $i = 1, \ldots, K-1$, by inductive hypothesis we have that:

$$\begin{aligned}
C_{K-1}(\boldsymbol{\eta}^{(1)})^{-1} &= \int_0^1\int_0^{1-x_1}\cdots\int_0^{1-x_1-\cdots-x_{K-3}} \exp\left(\sum_{i=1}^{K-2}(\eta_i-\eta_{K-1})x_i\right) dx_{K-2}\cdots dx_2 dx_1 \\
&= (-1)^K\sum_{k=1}^{K-1}\frac{\exp\left(\eta_k^{(1)}\right)}{\prod_{i\neq k}\left(\eta_i^{(1)}-\eta_k^{(1)}\right)} \\
&= (-1)^K\sum_{k=1}^{K-1}\frac{\exp\left(\eta_i-\eta_{K-1}\right)}{\prod_{i\neq k}\left(\eta_i-\eta_k\right)}. \qquad \text{(B.5)}
\end{aligned}$$

Similarly, letting $\eta_i^{(2)} = \eta_i$ for $i = 1, \ldots, K-2$, and $\eta_{K-1}^{(2)} = 0$, we have that:

$$\begin{aligned} C_{K-1}(\boldsymbol{\eta}^{(2)})^{-1} &= \int_0^1 \int_0^{1-x_1} \cdots \int_0^{1-x_1-\cdots-x_{K-3}} \exp\left(\sum_{i=1}^{K-2} \eta_i x_i\right) dx_{K-2} \cdots dx_2 dx_1 \\ &= (-1)^K \sum_{k=1}^{K-1} \frac{\exp\left(\eta_k^{(2)}\right)}{\prod_{i \neq k}\left(\eta_i^{(2)} - \eta_k^{(2)}\right)} \\ &= (-1)^K \left[\sum_{k=1}^{K-2} \frac{\exp(\eta_k)}{(-\eta_k)\prod_{i\neq k}(\eta_i - \eta_k)} + \frac{1}{\prod_{i=1}^{K-2}\eta_i}\right] \\ &= (-1)^K \left[\sum_{k=1}^{K-2} \frac{\exp(\eta_k)}{-(\eta_k - \eta_K)\prod_{i\neq k}(\eta_i - \eta_k)} + \frac{\exp(\eta_K)}{\prod_{i=1}^{K-2}(\eta_i - \eta_K)}\right]. \end{aligned} \tag{B.6}$$

Plugging (B.5) and (B.6) back into (B.4), we find:

$$C_K(\boldsymbol{\eta})^{-1} = (-1)^{K+1} \sum_{k=1}^{K} R_k(\boldsymbol{\eta}) \exp(\eta_k),$$

where the coefficients $R_k(\boldsymbol{\eta})$ gather the terms that multiply each $\exp(\eta_k)$ term. For $k = 1, \ldots, K-2$, both (B.5) and (B.6) contribute to the coefficient:

$$\begin{aligned} R_k(\boldsymbol{\eta}) &= \frac{1}{\eta_{K-1} - \eta_K}\left[-\frac{1}{\prod_{1\leq i\leq K-1, i\neq k}(\eta_i - \eta_k)} + \frac{1}{\prod_{1\leq i\leq K, i\neq k, i\neq K-1}(\eta_i - \eta_k)}\right] \\ &= \frac{1}{\eta_{K-1} - \eta_K}\left[\frac{-\eta_K + \eta_k + \eta_{K-1} - \eta_k}{\prod_{1\leq i\leq K, i\neq k}(\eta_i - \eta_k)}\right] \\ &= \frac{1}{\prod_{i\neq k}(\eta_i - \eta_k)}. \end{aligned} \tag{B.7}$$

The $(K-1)$th coefficient can be computed more easily as it only appears in (B.5):

$$\begin{aligned} R_{K-1}(\boldsymbol{\eta}) &= -\frac{1}{(\eta_{K-1} - \eta_K)} \frac{1}{\prod_{1\leq i\leq K-2}(\eta_i - \eta_{K-1})} \\ &= \frac{1}{\prod_{i\neq K-1}(\eta_i - \eta_{K-1})}, \end{aligned} \tag{B.8}$$

and similarly, the Kth coefficient appears only in (B.6):

$$R_K(\boldsymbol{\eta}) = \frac{1}{(\eta_{K-1} - \eta_K)} \frac{1}{\prod_{1 \le i \le K-2}(\eta_i - \eta_{K-1})}$$
$$= \frac{1}{\prod_{i \neq K}(\eta_i - \eta_K)}. \tag{B.9}$$

This completes the proof. □

Remark. For completeness, we also include the normalizing constant written in terms of the parameterization of the original density in equation 4.2 of the main text:

$$\int_{\mathbb{S}^{K-1}} \prod_{i=1}^{K} \lambda_i^{x_i} d\mu(\mathbf{x}) = (-1)^{K+1} \sum_{k=1}^{K} \frac{\lambda_k}{\prod_{i \neq k} \log \frac{\lambda_i}{\lambda_k}}.$$

B.2 Additional Properties of the CC Distribution

B.2.1 Mean and Covariance

As mentioned in the main manuscript (section 4.3.5), by standard properties of exponential families, the mean and covariance of the CC can be obtained by differentiating the normalizing constant. For completeness, we include these results here. If $\mathbf{x} \sim CC(\boldsymbol{\eta})$, then the mean of $\mathbf{x}$ is given by:

$$\mathbb{E}[x_i] = -\frac{\partial}{\partial \eta_i} \log C(\boldsymbol{\eta}), \tag{B.10}$$

and the covariance is given by:

$$\text{cov}(x_i, x_j) = -\frac{\partial^2}{\partial \eta_i \partial \eta_j} \log C(\boldsymbol{\eta}). \tag{B.11}$$

B.2.2 KL Divergence

The KL divergence between two CC variates can be computed directly from their means:

$$\begin{aligned} KL(p(\mathbf{x}|\boldsymbol{\eta})||p(\mathbf{x}|\tilde{\boldsymbol{\eta}})) &= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})}\left[\log \frac{p(\mathbf{x}|\boldsymbol{\eta})}{p(\mathbf{x}|\tilde{\boldsymbol{\eta}})}\right] \\ &= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})}\left[\log C(\boldsymbol{\eta}) - \log C(\tilde{\boldsymbol{\eta}}) + \sum_{i=1}^{K-1}(\eta_i - \tilde{\eta}_i)x_i\right] \\ &= \log C(\boldsymbol{\eta}) - \log C(\tilde{\boldsymbol{\eta}}) + (\boldsymbol{\eta} - \tilde{\boldsymbol{\eta}})^\top \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\eta})}[\mathbf{x}]. \end{aligned} \tag{B.12}$$

B.2.3 Moment Generating Function

The moment generating function of the CC distribution can be written directly in terms of the normalizing constant:

$$\begin{aligned} M_{\mathbf{x}}(\mathbf{t}) &= \mathbb{E}[e^{\mathbf{t}^\top \mathbf{x}}] \\ &= \int_{\mathbb{S}^{K-1}} e^{\mathbf{t}^\top \mathbf{x}} C(\boldsymbol{\eta}) e^{\boldsymbol{\eta}^\top \mathbf{x}} d\mu \\ &= C(\boldsymbol{\eta}) \int_{\mathbb{S}^{K-1}} e^{(\mathbf{t}+\boldsymbol{\eta})^\top \mathbf{x}} d\mu \\ &= \frac{C(\boldsymbol{\eta})}{C(\mathbf{t}+\boldsymbol{\eta})}. \end{aligned} \tag{B.13}$$

The characteristic function can be derived similarly.

B.2.4 Marginalization

Unlike the Dirichlet, the CC is not preserved under marginalization, even when allowing transformations of the parameter vector. In other words, if $(x_1, \ldots, x_{K-1}) \sim CC(\eta_1, \ldots, \eta_{K-1})$, then it is not true that $x_1 \sim CC(\eta_1)$, nor that $(x_1, \ldots, x_{K-2}) \sim CC(\eta_1, \ldots, \eta_{K-2})$. It is not even true that $x_1 \sim CC(\tilde{\eta}_1)$, nor that $(x_1, \ldots, x_{K-2}) \sim CC(\tilde{\eta}_1, \ldots, \tilde{\eta}_{K-1})$, for any $\tilde{\boldsymbol{\eta}}$. This can be seen easily by

integrating out the case $K = 3$:

$$\begin{aligned}\int_0^{1-x_1} C(\eta_1,\eta_2)\exp(\eta_1 x_1 + \eta_2 x_2)dx_2 &= C(\eta_1,\eta_2)\exp(\eta_1 x_1)\left[\frac{\exp(\eta_2 t)}{\eta_2}\right]_{t=0}^{t=1-x_1} \\ &= \frac{C(\eta_1,\eta_2)\exp(\eta_2)}{\eta_2}\exp((\eta_1-\eta_2)x_1) - \frac{C(\eta_1,\eta_2)}{\eta_2}\exp(\eta_1 x_1),\end{aligned} \tag{B.14}$$

which is not of the form $C(\tilde{\eta}_1)\exp(\tilde{\eta}_1 x_1)$ for any $\tilde{\eta}_1$.

As a direct consequence, we cannot use a stick-breaking construction (Connor and Mosimann, 1969; Paisley et al., 2010) to simulate CC variates from 1-dimensional CB variates, as with the Dirichlet and the Beta distributions.

B.3 Sampling

In this section, we develop sampling algorithms for the CC distribution and analyze their performance empirically. We also describe how to use our samplers to obtain reparameterization gradients (Kingma and Welling, 2014).

B.3.1 The 'Naive' Rejection Sampler

Given the form of the CC density function, a rejection sampling scheme follows readily by combining independent 1-dimensional CB draws (algorithm 4).

Algorithm 4 Naive sampler

Input: target distribution $CC(\boldsymbol{\lambda})$.
Output: sample $\mathbf{x}$ drawn from target.

1: For $i = 1,\ldots,K-1$, draw $x_i \sim CC(\lambda_i,\lambda_K)$ independently.
2: If $\sum_{i=1}^{K-1} x_i > 1$, go back to step 1, otherwise return $\mathbf{x} = (x_1,\ldots,x_{K-1})$.

To see why algorithm 4 achieves the desired distribution, firstly note that by independence, the

distribution produced in step 1 is:

$$p_{\text{step1}}(\mathbf{x}) \propto \prod_{i=1}^{K-1} \lambda_i^{x_i} \lambda_K^{1-x_i} \propto \lambda_1^{x_1} \cdots \lambda_{K-1}^{x_{K-1}} \lambda_K^{1-x_1-\cdots-x_{K-1}}. \tag{B.15}$$

This is precisely the density we seek, except it is drawn on $[0,1]^{K-1}$ instead of the simplex. Step 2 rejects all samples that fall outside the simplex, thus achieving the target distribution.

The obvious shortcoming of this sampling approach is that, even for moderate values of K, the proportion of rejections becomes large. This is particularly troublesome in the balanced case, $\mathbf{x} \sim CC(1/K, \ldots, 1/K)$, which is equivalent to drawing uniformly on $[0,1]^{K-1}$ and rejecting whenever we fall outside of a simplex of measure $1/(K-1)!$. In other words, we accept with a probability that decays factorially in dimension.

Reparameterization

The 1-dimensional CB distribution can be reparameterized using the analytical expression for the inverse CDF, derived by Loaiza-Ganem and Cunningham (2019). In this section we extend the strategy to a multivariate analogue for the CC distribution. The underlying idea is that the rejection step in algorithm 4 only depends on the L_1 norm of the proposal, but not on the parameter. This implies that, once we find an accepted proposal, we can use the inverse CDF reparameterization directly, without requiring a correction term as per the general framework for acceptance-rejection reparameterization gradients (Naesseth et al., 2017).

Our aim is to write $\mathbf{x} = g(\mathbf{u}, \boldsymbol{\lambda})$, where the density of $\mathbf{u}$ does not depend on $\boldsymbol{\lambda}$. To this end, write $F(x|\lambda_i, \lambda_K)$ for the CDF of $x \sim CC(\lambda_i, \lambda_K)$. Note that this expression will follow readily from an equivalent CB distribution, as $CC(\lambda_i, \lambda_K) = C\mathcal{B}(\lambda_i/(\lambda_i + \lambda_K))$. For each $i = 1, \ldots, K-1$, applying the inverse CDF component-wise on each of $u_i \overset{iid}{\sim} U(0,1)$ results in $F^{-1}(u_i|\lambda_i, \lambda_K) \sim CC(\lambda_i, \lambda_K)$. Thus, the vector

$$\mathbf{F}^{-1}(\mathbf{u}|\boldsymbol{\lambda}) := [F^{-1}(u_1|\lambda_1, \lambda_K), \ldots, F^{-1}(u_{K-1}|\lambda_{K-1}, \lambda_K)] \tag{B.16}$$

provides a differentiable reparameterization of the distribution $CC(\boldsymbol{\lambda})$, provided $\mathbf{u}$ was drawn from the pre-image of the simplex $\mathbb{S}^{K-1}$ under the mapping $\mathbf{F}^{-1}$, or in other words, provided that $\mathbf{u} \in \mathbf{F}(\mathbb{S}^{K-1})$. The rejection step simply guarantees that we find a sample of uniforms inside this region, but once we have found such a sample, it will lie in the interior of the region with probability 1, and therefore we can differentiate through the transformation as desired:

$$\frac{\partial \mathbf{x}}{\partial \boldsymbol{\lambda}} = \frac{\partial}{\partial \boldsymbol{\lambda}} \mathbf{F}^{-1}(\mathbf{u}|\boldsymbol{\lambda}). \tag{B.17}$$

We formalize this reparameterization in algorithm 5.

Algorithm 5 Reparameterized rejection sampler

Input: target distribution $CC(\boldsymbol{\lambda})$.
Output: a sample $\mathbf{u}$ such that $\mathbf{F}^{-1}(\mathbf{u}|\boldsymbol{\lambda}) \sim CC(\boldsymbol{\lambda})$.

1: For $i = 1, \ldots, K-1$, draw $u_i \sim U(0,1)$ and set $x_i = F^{-1}(x|\lambda_i, \lambda_K)$.
2: If $\sum_{i=1}^{K} x_i > 1$, return to step 1, otherwise return $\mathbf{u} = (u_1, \ldots, u_{K-1})$.

B.3.2 The Ordered Rejection Sampler

An analysis of algorithm 4 reveals two relevant observations. Firstly, the simulation of each $x_j \sim CC(\lambda_j, \lambda_K)$ in step 1 involves computing the inverse cdf $F^{-1}(\cdot|\lambda_i, \lambda_K)$, which is more expensive than computing the cumulative sum $\sum_{i=1}^{j} x_i$ of the draws. It therefore pays to recompute the cumulative sums after each draw and go directly to the rejection step as soon as it exceeds 1. Secondly, note that we do not generally expect the components of $\boldsymbol{\lambda}$ to be balanced. Thus, even though simulating each x_i in step 1 requires the same amount of computation, those drawn from smaller values of λ_i are more likely to be close to 0 than those drawn from higher values of λ_i. The dimensions that are more likely to be close to 0 are also less likely to make our cumulative sum exceed the rejection threshold. It therefore also pays to draw the x_i components in order of decreasing λ_i.

These remarks motivate an improved sampling scheme, which we call the ordered rejection sampler (algorithm 6). Empirically, we find that this sampler substantially reduces the rejection

rate (see figure B.1) as well as the computation time (by not only rejecting less, but also rejecting sooner). However, this sampler performs poorly when $\boldsymbol{\lambda}$ is balanced; such a setting leaves little room for improvement from the re-ordering operation, and the resulting sampler is similar to the naive rejection sampler. This motivates a further sampling scheme that we introduce in the following section, but further improvements to this sampler are left to future work. Lastly, we note that the reparameterization scheme of section B.3.1 can be modified trivially to apply here also.

Algorithm 6 Ordered rejection sampler

Input: target distribution $CC(\boldsymbol{\lambda})$.
Output: sample $\mathbf{x}$ drawn from target.

1: Find the permutation π that orders $\boldsymbol{\lambda}$ from largest to smallest, and let $\tilde{\boldsymbol{\lambda}} = \pi(\boldsymbol{\lambda})$.
2: Set the cumulative sum $c \leftarrow 0$ and $i \leftarrow 2$.
3: **while** $c < 1$ **do**
4: Draw $u_i \sim U(0, 1)$.
5: Set $x_i = F^{-1}(u_i|\tilde{\lambda}_i, \tilde{\lambda}_1)$.
6: Set $c \leftarrow c + x_i$.
7: Set $i \leftarrow i + 1$.
8: **end while**
9: If $c > 1$, go back to step 2.
10: Set $x_1 = 1 - \sum_{i=2}^{K} x_i$.
11: Return $\mathbf{x} = \pi^{-1}(x_1, \ldots, x_K)$.

B.3.3 The Permutation Sampler

Next, we develop a permutation sampler that performs particularly well for configurations of $\boldsymbol{\lambda}$ that are balanced (those that lead to distributions that are close to uniform). Our key insights here are that the unit cube can be partitioned into simplexes, each of which corresponds to a permutation of its dimensions, and that the CC distribution is, in a sense, 'invariant' over these permutations.

Partitioning the cube into simplexes

Let $\mathcal{R} = [0, 1]^{K-1}$, the unit cube. For a permutation $\sigma : \{1, 2, \ldots, K-1\} \rightarrow \{1, 2, \ldots, K-1\}$, we denote $\mathcal{S}_\sigma = \{\mathbf{x} \in \mathbb{R}^{K-1} : 0 \leq x_{\sigma(1)} \leq x_{\sigma(2)} \leq \cdots \leq x_{\sigma(K-1)} \leq 1\}$. We can then partition (up to

intersections of Lebesgue measure zero) the cube using the $(K-1)!$ different permutations:

$$\mathcal{R} = \bigcup_{\sigma} \mathcal{S}_{\sigma}, \tag{B.18}$$

where the union is over all permutations. While our sample space $\text{cl}(\mathbb{S}^{K-1})$, is not equal to $\mathcal{S}_{\sigma}$ for any σ, we will see in section B.3.3 that sampling from $\text{cl}(\mathbb{S}^{K-1})$ and sampling from $\mathcal{S}_{id}$ are equivalent, where id is the identity permutation. However, as we will see in section B.3.3, sampling from $\mathcal{S}_{id}$ allows to take advantage of the cube partitioning of equation B.18, while the same cannot be done for $\text{cl}(\mathbb{S}^{K-1})$.

The equivalence of sampling over any simplex

In this section, we consider varying the support of our CC density from the standard simplex, to other simplexes as well as the unit cube. We denote the support explicitly by writing $\mathbf{x} \sim CC_{\mathcal{A}}(\boldsymbol{\eta})$ for the density:

$$p_{\mathcal{A}}(\mathbf{x}|\boldsymbol{\eta}) \propto \exp\left(\sum_{i=1}^{K-1} \eta_i x_i\right) \mathbb{1}(\mathbf{x} \in \mathcal{A}) \tag{B.19}$$

where the subscript $\mathcal{A}$ will typically denote a simplex. Now, letting $\mathbf{x} \sim CC_{\mathcal{A}}(\boldsymbol{\eta})$ and $\mathbf{y} = Q\mathbf{x}$, where $Q \in \mathbb{R}^{(K-1)\times(K-1)}$ is an invertible matrix, it follows by the change of variable formula that:

$$\begin{aligned} p_{Q(\mathcal{A})}(\mathbf{y}|\boldsymbol{\eta}) &= \frac{1}{|\det(Q)|} p_{\mathcal{A}}(Q^{-1}\mathbf{y}|\boldsymbol{\eta}) \\ &\propto \exp(\boldsymbol{\eta}^{\top}[Q^{-1}\mathbf{y}])\mathbb{1}(y \in Q(\mathcal{A})) \\ &= \exp([Q^{-\top}\boldsymbol{\eta}]^{\top}\mathbf{y})\mathbb{1}(y \in Q(\mathcal{A})), \end{aligned} \tag{B.20}$$

where $Q(\mathcal{A}) = \{\mathbf{y} : \mathbf{y} = Q\mathbf{x}, \mathbf{x} \in \mathcal{A}\}$. Thus, we have that $\mathbf{y} \sim CC_{Q(\mathcal{A})}(\tilde{\boldsymbol{\eta}})$, where $\tilde{\boldsymbol{\eta}} = Q^{-\top}\boldsymbol{\eta}$, so that $\mathbf{y}$ has a new CC distribution on a transformed sample space. Moreover, if Q is a permutation matrix and $\mathcal{A} = \mathcal{S}_{\sigma}$ for some permutation σ, then $Q(\mathcal{A})$ is a 'permuted' simplex, and $\tilde{\boldsymbol{\eta}}$ is a rearranged parameter vector, hence the equivalence of sampling over any simplex for the CC.

The permutation sampling algorithm

Now, consider a lower triangular matrix of ones:

$$B = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix}. \tag{B.21}$$

Note that $\mathcal{S}_{id} = B(\text{cl}(\mathbb{S}^{K-1}))$, so that sampling from $CC_{\text{cl}(\mathbb{S}^{K-1})}(\boldsymbol{\eta})$ is equivalent to sampling from $CC_{\mathcal{S}_{id}}(\tilde{\boldsymbol{\eta}})$ and transforming the result with B^{-1}, where $\tilde{\boldsymbol{\eta}} = B^{-\top}\boldsymbol{\eta}$. Now consider rejection sampling to draw from $CC_{\mathcal{S}_{id}}(\tilde{\boldsymbol{\eta}})$. As with the naive sampler, our proposal can be drawn on the whole unit cube from independent 1-dimensional CB variates, but the advantage here is that we do not have to directly reject the sample if it fell outside of the desired simplex $\mathcal{S}_{id}$, but rather we can transform it onto that simplex and then accept it with an appropriate probability (which we can compute easily using the invariance property). Here, the acceptance probability depends on which simplex the proposal fell into, and is given by:

$$\alpha(\mathbf{y}, \tilde{\boldsymbol{\eta}}, P) = \frac{p_{\mathcal{S}_{id}}(\mathbf{y}|\tilde{\boldsymbol{\eta}})}{\kappa(\tilde{\boldsymbol{\eta}}, P) p_{\mathcal{S}_{id}}(\mathbf{y}|P^{-\top}\tilde{\boldsymbol{\eta}})}, \tag{B.22}$$

where $\kappa(\tilde{\boldsymbol{\eta}}, P)$ is the rejection sampling constant, which in this case is equal to:

$$\kappa(\tilde{\boldsymbol{\eta}}, P) = \max_{\mathbf{y} \in \mathcal{S}_{id}} \frac{p_{\mathcal{S}_{id}}(\mathbf{y}|\tilde{\boldsymbol{\eta}})}{p_{\mathcal{S}_{id}}(\mathbf{y}|P^{-\top}\tilde{\boldsymbol{\eta}})}. \tag{B.23}$$

The algorithm samples correctly from $CC_{\mathcal{S}_{id}}(\tilde{\boldsymbol{\eta}})$, because $\mathbf{y}'$ can be thought of as a sample from $p_{\mathcal{R}}(\mathbf{y}'|\boldsymbol{\eta}, \mathbf{y}' \in \mathcal{S}_\sigma) = p_{\mathcal{S}_\sigma}(\mathbf{y}'|\boldsymbol{\eta})$, which we then transform with P to obtain a distribution on $\mathcal{S}_{id}$. If we use this distribution as a proposal distribution for a rejection sampling algorithm, we recover precisely the acceptance probability of equation B.22. Intuitively, if our sample $\mathbf{x}$ does not fall on

Algorithm 7 Permutation sampler

Input: target distribution $CC(\boldsymbol{\eta})$.
Output: sample $\mathbf{x}$ drawn from target.

1: Sample $\mathbf{y}' \sim CC_{\mathcal{R}}(\tilde{\boldsymbol{\eta}})$ (again, this is straightforward to do by sampling each coordinate independently).
2: Sort the elements of $\mathbf{y}'$. In other words, find a permutation σ such that $\sigma(\mathbf{y}') \in \mathcal{S}_{id}$. Let P be the corresponding permutation matrix and $\mathbf{y} = P\mathbf{y}'$.
3: Compute $\kappa(\tilde{\boldsymbol{\eta}}, P)$ by taking the maximum over the the vertices of $\mathcal{S}_{id}$, and use this to compute the acceptance probability $\alpha(\mathbf{y}, \tilde{\boldsymbol{\eta}}, P)$.
4: Accept $\mathbf{y}$ with probability $\alpha(\mathbf{y}, \tilde{\boldsymbol{\eta}}, P)$. Otherwise, go back to step 1.
5: Return $\mathbf{x} = B^{-1}\mathbf{y}$.

the desired simplex $\mathcal{S}_{id}$, we move around the simplex in which it fell (along with $\mathbf{x}$ itself) so that it matches the desired simplex, and then do rejection sampling.

We conclude this subsection with two short notes on the optimization problem of equation B.23. The first one is that when $\boldsymbol{\eta}^{(2)} = Q^{-\top}\boldsymbol{\eta}^{(1)}$ where $|\det(Q)| = 1$, then the normalizing constants cancel out, which is the case in our algorithm since $|\det(P)| = 1$. The second is that, by taking logs, the optimization problem can be transformed into a linear problem subject to linear inequality constraints, meaning that the solution must be achieved at a vertex. Since there are K vertices, namely $\mathbf{0}$, $\mathbf{e}_{K-1}, \mathbf{e}_{K-1} + \mathbf{e}_{K-2}, \ldots, \sum_{i=1}^{K-1} \mathbf{e}_i$, we can solve the problem by simply checking each of these vertices.

B.3.4 Performance

While the ordered rejection sampler can never have a worse rejection rate than its naive counterpart, the comparison with the permutation sampler depends on the shape of the target distribution, as discussed. The perfectly balanced case $\boldsymbol{\lambda} = (1/K, \ldots, 1/K)$ results in the worst possible rejection rate for the ordered rejection sampler (we accept with probability $1/(K-1)!$), but also the best possible rejection rate for the permutation sampler (this is the uniform case so $\alpha(\mathbf{y}, \tilde{\boldsymbol{\eta}}, P) = 1$). On the other end of the spectrum, in the totally unbalanced case where one element of $\boldsymbol{\lambda}$ holds all the weight and the others are close to zero, the ordered rejection sampler achieves an acceptance rate close to 1, whereas it is much smaller for the permutation sampler (see section B.3.4).

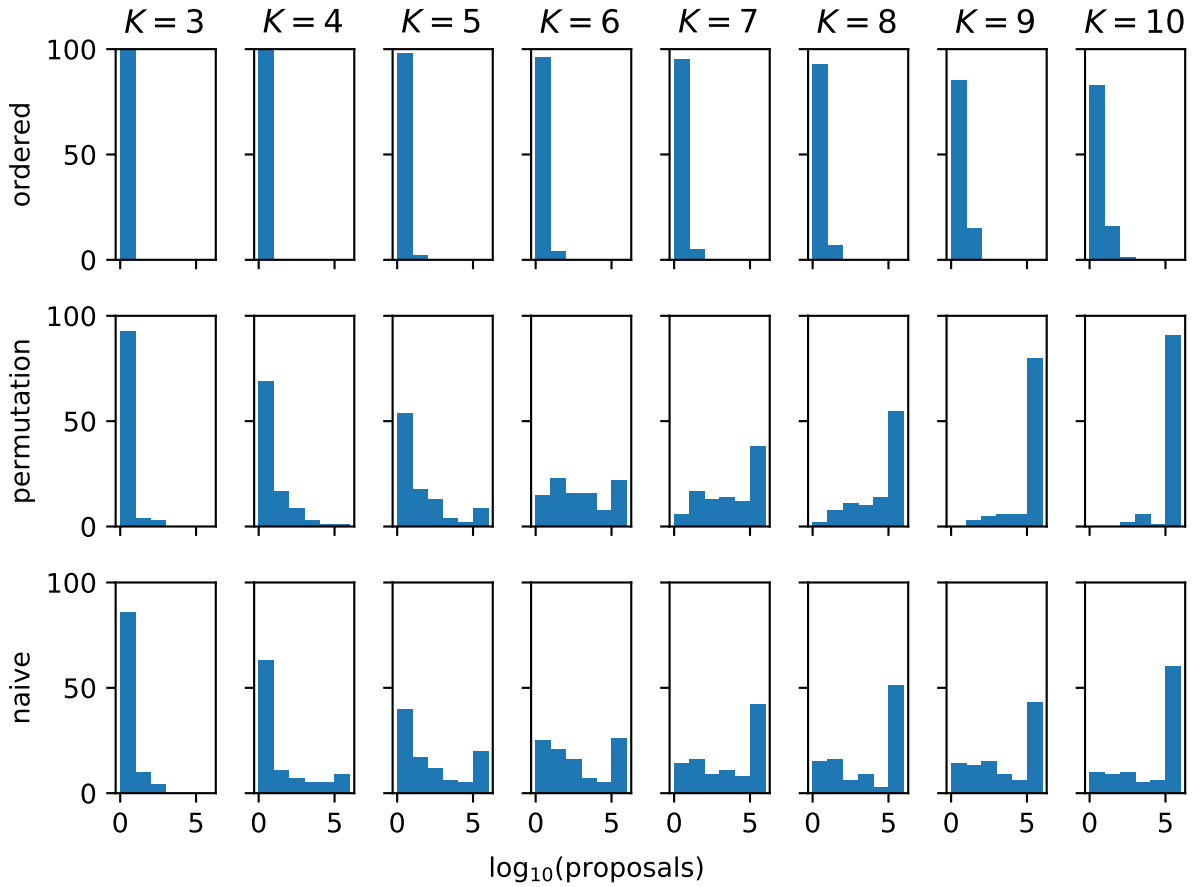


Figure B.1: Shows the performance of 3 sampling algorithms across different dimensions K. Each histogram shows the distribution, over 100 trials, of the number of proposals required for 1 acceptance, on the log scale (base 10). The distributions are not exponential, since each of the 100 trials is sampled from a different $CC(\boldsymbol{\lambda})$ distribution, where the parameter follows independent and identically distributed $\boldsymbol{\lambda} \sim Dirichlet(1/K, \cdots, 1/K)$. Due to computational constraints, the number of proposals in each trial is right-censored, hence the large bars at the right end of the histograms.

In this sense, our samplers are complementary, and an optimal sampling algorithm could involve combining accept/reject steps from both methods. We study the performance of our samplers empirically, by comparing the distribution of the rejection rates under a sparsity-inducing prior $\boldsymbol{\lambda} \sim Dirichlet(1/K, \ldots, 1/K)$. Indeed, the ordered rejection sampler tends to considerably outperform the permutation sampler (see figure B.1), as well as (trivially) the naive sampler.

Now, it may come as a surprise that the permutation sampler does not necessarily outperform the naive sampler. After all, the naive sampler only accepts samples that fell directly into the desired simplex, whereas the permutation sampler has the additional possibility of accepting a

sample that fell outside of $\mathcal{S}_{id}$ after applying a suitable permutation. However, this intuition breaks down once we realize that the proposal distributions from the two methods are not equivalent. We make this precise in the following sections.

Rejection rate - naive sampler

Suppose we seek $\mathbf{x} \sim CC_{\mathrm{cl}(\mathbb{S}^{K-1})}(\boldsymbol{\eta})$. The naive rejection sampler proposes $\mathbf{x} \sim CC_{\mathcal{R}}(\boldsymbol{\eta})$, and accepts if $\mathbf{x} \in \mathbb{S}^{K-1}$. The proposal density is equal to (we know the normalizing constant as we have the product of independent CBs):

$$p_{\mathcal{R}}(\mathbf{x}|\boldsymbol{\eta}) = \prod_{i=1}^{K-1} \frac{\eta_i}{e^{\eta_i} - 1} e^{\eta_i x_i}. \tag{B.24}$$

Therefore, the probability of acceptance is:

$$P(CC_{\mathcal{R}}(\boldsymbol{\eta}) \in \mathbb{S}^{K-1}) = \int_{\mathbb{S}^{K-1}} \prod_{i=1}^{K-1} \frac{\eta_i}{e^{\eta_i} - 1} e^{\eta_i x_i} d\mu. \tag{B.25}$$

We can apply the transformation B to rewrite this as:

$$P(B(CC_{\mathcal{R}}(\boldsymbol{\eta})) \in B(\mathbb{S}^{K-1})) = P(CC_{B(\mathcal{R})}(B^{-\top}\boldsymbol{\eta}) \in S_{id}) = \int_{\mathcal{S}_{id}} \prod_{i=1}^{K-1} \frac{\eta_i}{e^{\eta_i} - 1} e^{\tilde{\eta}_i x_i} d\mu, \tag{B.26}$$

where we have used the fact that $|\det(B)| = 1$ so that the normalizing constant remains unchanged. Thus, the probability of acceptance of the naive sampler is equal to:

$$P(accept) = \left(\prod_{i=1}^{K-1} \frac{\eta_i}{e^{\eta_i} - 1} \right) \cdot \int_{\mathcal{S}_{id}} e^{\tilde{\boldsymbol{\eta}}^{\top}\mathbf{x}} d\mu. \tag{B.27}$$

Rejection rate - permutation sampler

In the case of the permutation sampler, the acceptance rate is harder to compute. However, one can easily obtain a lower bound by considering only the samples that fall directly into our target

simplex (in this case, $\mathcal{S}_{id}$). In this case, the proposal distribution is:

$$p_{\mathcal{R}}(\mathbf{x}|\tilde{\boldsymbol{\eta}}) = \prod_{i=1}^{K-1} \frac{\tilde{\eta}_i}{e^{\tilde{\eta}_i} - 1} e^{\tilde{\eta}_i x_i}. \tag{B.28}$$

If the resulting sample falls in $\mathcal{S}_{id}$, we accept the sample and map it back to $\text{cl}(\mathbb{S}^{K-1})$. Thus, the acceptance rate has the lower bound

$$P(accept) \geq \left(\prod_{i=1}^{K-1} \frac{\tilde{\eta}_i}{e^{\tilde{\eta}_i} - 1} \right) \cdot \int_{\mathcal{S}_{id}} e^{\tilde{\boldsymbol{\eta}}^\top \mathbf{x}} d\mu. \tag{B.29}$$

Note that, while this lower bound has the same integral term as the naive rejection sampler, it is multiplied by a different normalizing constant. In particular, there are configurations of $\boldsymbol{\eta}$ that can lead to much worse normalizing constants for the permutation sampler than for the naive rejection sampler, resulting in a worse acceptance rate overall.

Example

We give an example of a configuration of $\boldsymbol{\eta}$ such that the acceptance rate of the naive sampler is better than that of the permutation sampler. Consider the case $(\eta_1, \cdots, \eta_{K-1}) = (-M, \cdots, -M)$, where M is a large positive number. Note that this example is far from the uniform case $\boldsymbol{\lambda} = (1/K, \cdots, 1/K)$. In fact, in this case, after transforming with B, we obtain $\tilde{\eta}_{K-1} = -M$, and $\tilde{\eta}_{K-2} = \cdots = \tilde{\eta}_1 = 0$. Thus, when we sample a proposal $\mathbf{y} \sim CC_{\mathcal{R}}(\tilde{\boldsymbol{\eta}})$, typically y_{K-1} will be small relative to $y_1, \cdots, y_{K-2}$, which will be ordered at random. This means the sorting step of our permutation sampler will likely map y_{K-1} to y'_1, as well as sorting the remaining entries $y'_2 < \cdots < y'_{K-1}$. In other words, P maps the $(K-1)^{th}$ entry to the 1st entry, and one of the first $K-2$ entries to the $(K-1)^{th}$ entry (whichever of these happens to sample the largest value). The resulting distributions $p_{\mathcal{S}_{id}}(\cdot|\tilde{\boldsymbol{\eta}})$ and $p_{\mathcal{S}_{id}}(\cdot|P^{-\top}\tilde{\boldsymbol{\eta}})$ are similar, with the key difference that the former puts the negative $\tilde{\boldsymbol{\eta}}$ coefficient (namely $\tilde{\eta}_{K-1} = -M$) in the last position (the largest component of $\mathbf{y}$), while the latter puts it into some other position determined by σ^{-1}, i.e. the right-most column of P or equivalently, the bottom row of P^{-1}. In this setting, it follows that κ will be equal to 1,

and the ratio $p_{\mathcal{S}_{id}}(\mathbf{y}'|\tilde{\boldsymbol{\eta}})/p_{\mathcal{S}_{id}}(\mathbf{y}'|P^{-\top}\tilde{\boldsymbol{\eta}})$ will be small. Thus, our rejection sampling ratio is typically small and we are likely to reject our proposal. The ratio will be close to 1 only in the event that the proposed value x_{K-1} is large relative to the other components $x_1, \cdots, x_{K-2}$, which rarely happens as x_{K-1} is sampled from a univariate CC with much smaller coefficient. Note further, that $\tilde{\boldsymbol{\eta}}$ cannot be re-shuffled in this case, as this would lead to a target distribution on a simplex other than $\mathcal{S}_{id}$ (we can only shuffle $\boldsymbol{\eta}$ prior to applying B, which in this case leaves $\boldsymbol{\eta}$ unchanged). We conclude that we cannot achieve a uniformly better rejection rate through the permutation sampler, relative to the naive method.

Appendix C: Uses and Abuses of the Cross Entropy Loss

C.1 Experimental details

C.1.1 Label Smoothing

We denote a convolutional layer by W×W×N-S, where W is the width of the convolution, N the number of filter maps, and S the stride. Our architecture is: 3×3×32-1 → BatchNorm → 3×3×32-1 → BatchNorm → MaxPooling (2×2) → Dropout (0.2) → 3×3×64-1 → BatchNorm → 3×3×64-1 → BatchNorm → MaxPooling (2×2) → Dropout (0.3) → 3×3×128-1 → BatchNorm → 3×3×128-1 → BatchNorm → MaxPooling (2×2) → Dropout (0.4) → 10 fully-connected units. We use weight decay of 0.0001 in the final fully-connected layer, which, together with dropout and batch normalization, was switched on and off in the different runs of our ablation study in Table 5.1. Our models were trained for 500 epochs using a minibatch size of 128 and the Adam optimizer with a learning rate of 10^{-3}. The label smoothing hyperparameter ε was set to 0.1 as per (Müller, Kornblith, and Hinton, 2019).

Note however, that we were unable to replicate the results of (Müller, Kornblith, and Hinton, 2019) exactly, as they did not share their code, nor did they describe their architecture in full.

C.1.2 Actor-Mimic Network

Our architecture is 8×8×32-4 → 4×4×64-2 → 3×3×64-1 → 7×7×1024-1 → 512 fully-connected units → 6 fully connected units (corresponding to 6 possible actions). We used the Adam optimizer with a learning rate of 10^{-5}, and a minibatch size of 32.