An end to end Machine Learning Model for Compositional Data

Sean Lamont

A thesis submitted for the degree of
Bachelor of Advanced Computing (Research and Development) (Honours)
The Australian National University

October 2019

Except where otherwise indicated, this thesis is my own original work.

Sean Lamont
24 October 2019

to my Brunhilde

Acknowledgments

I would firstly like to thank my supervisors Dr. Cheng Soon Ong and Dr. Christian Walder for their assistance and helpful suggestions over the course of this project. Next I want to thank Chamin Hewa for lending me his encyclopedic knowledge on all things Machine Learning throughout the project. I'd also like to thank Chris Williams for his feedback on drafting this thesis.

Abstract

Compositional Data Analysis (CoDA) is the study of data which can be interpreted as ratios or counts. Data of this type manifests in many forms, ranging from microbiome species counts to wealth distributions. Mathematically, compositional data is known to exist in a simplex. This prevents many standard analysis methods, which assume Euclidean geometry, from being applied. Dimensionality reduction through Principal Component Analysis (PCA) is once such method. There are instead approaches to dimensionality reduction which are faithful to this compositional structure. These methods have been shown to preserve structures within the original data which are distorted by PCA.

This thesis investigates representations of compositional data, and how they influence supervised learning. Although there are visual examples of how PCA does not preserve compositional structure, it is not certain how this representation influences performance when applied to supervised learning. By the same token, we do not know whether mathematically justified methods can improve performance. We compare standard Euclidean representations to those given by Compositional Data Analysis (CoDA) in terms of how they impact subsequent supervised learning performance. Doing this on three microbiome datasets, we found a significant improvement in performance when supervised learning was preceded by the CoDA methods.

Our main contribution is the introduction of a novel end-to-end Machine Learning model. This model constructs a representation as a function of both a reconstruction and supervised loss. When testing this model on a smaller subset of the same data, we found it to give representations which reveal more useful structures for supervised learning. In these cases, this model outperformed all baseline methods including the CoDA based algorithms. For large dimensions, this model does not perform as well, which we argue is due to the relatively small sample sizes.

Our findings add to the growing body of literature which shows the importance of properly justified Compositional Data Analysis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Machine Learning (ML) has greatly increased the ability to discover patterns in data with modern computational power. Today ML is ubiquitous, as it can predict outcomes and extract insights from data which would not otherwise be possible. Generally speaking, most ML problems can be divided into one of two categories. The first is supervised learning. Here we have a set of inputs (the features) which are used to predict an outcome (the target). Speech recognition, stock market predictions and disease identification are all examples of this. Unsupervised learning is the other category. In this paradigm, we focus on the analysis of data without targets. Dimensionality reduction is a key example of this, and refers to the methods used to represent data in a lower dimensional space. Doing so can remove redundant information, improve efficiency and allow for the visualisation of high dimensional data.

Most ML algorithms make only a few assumptions on the data, hence their application to such broad settings. This being said, there are certain cases where these assumptions do not hold. One such case is Compositional Data, for which data is represented as relative proportions. Here the absolute value of the data only matters in how it relates to the other components. For example, in geology the amount of each element in a rock composition only matters in relation to the other components. A rock with 2g of iron and 4g of nickel would form the same composition as a rock with 4g of iron and 8g of nickel. Any data which involves percentages or counts summing to a constant falls in this category, and so the scope of such data is vast. Economic data (e.g. GDP and wealth distribution), chemical compositions, and geology (soil or rock compositions) are just a few applications.

Microbiome studies are a particularly important example of Compositional Data. These data represent the distribution of microorganisms within a target site, such as the human gut. The application of supervised learning to microbiome studies covers a range of applications. There is the potential to improve human health through studying the organisms present in our stomachs or stool. We can use the frequencies of these organisms as features, and the information surrounding them as targets in a supervised learning model. Details of some examples where this has already been done

can be found in Qu et al. [2019]. In this paper, the first example is disease prediction, where we can use microbiome data as biomarkers to detect diseases which may not otherwise have been noticed. Different people respond differently to the same diets or medicine, and we know that host specific phenotypes (i.e. observable characteristics) can be predicted through supervised learning [Qu et al., 2019]. We could therefore use an individuals microbiome profile to customise a diet or treatment plan unique to their needs. One interesting application beyond the human microbiome was found in a recent Nature study on cattle and sheep [Stewart et al., 2019]. Here, they presented results which clearly distinguished high and low methane emitting sheep based on their microbiome. They discuss the potential for manipulating these sheep to reduce methane output. If successful, this would contribute to a significant reduction in global emissions.

The examples above were made possible through the application of supervised learning models. It is important that these models are designed with the compositional structure of the data in mind. The standard approach to this was introduced by Aitchison [1982], through log ratio coordinates. Applying this Aitchinson transformation allows standard methods to be applied directly to the new coordinate space. Recent research [Avalos et al., 2018] has also given us a new algorithm for dimensionality reduction of compositional data, called CoDA-PCA.

1.2 This Thesis

Compositional data does not satisfy the standard assumptions required for most ML algorithms. Despite this, it remains common practice to apply these methods to such data, violating the theoretical assumptions created for these tools. This is especially true for microbiome studies, with Gloor et al. [2017] noting that they often ignore the inherent compositional structure. Standard Principal Component Analysis (PCA) is often the algorithm of choice when it comes to dimensionality reduction in these cases, despite the associated problems (see e.g. Stewart et al. [2019], McDonald et al. [2018]). It is well known that PCA representations do not preserve the compositional structure of the data [Avalos et al., 2018]. What is less clear is whether this distortion of structure negatively impacts the performance when PCA is used prior to supervised learning. Conversely, it is unclear whether methods for dimensionality reduction which account for this structure can improve performance. We know through data visualisation that these methods are more faithful to certain structures in the data (for example, CoDA-PCA Avalos et al. [2018]). This does not imply that their representations lead to better predictive performance. For this thesis, we investigate the impact of different representation methods on supervised learning performance. In doing so, we provide an objective measure which compares these approaches, in contrast to the somewhat subjective notion of visually preserving structure. Specifically, our results focus on two questions.

- Do CoDA based representations improve predictive performance in comparison to traditional (PCA) representations and direct supervised learning?

As mentioned above, CoDA based methods (i.e. Aitchison [2003] and CoDA-PCA [Avalos et al., 2018]) can visually preserve more structure within the data. In answering this question we can provide another, arguably more objective measure to evaluate their performance. A positive answer to this question would support the importance of properly justified CoDA, as it would therefore improve supervised learning on compositional data.

- Can end-to-end learning give better representations than baseline methods, as judged by the supervised performance?

Dimensionality reduction does not usually consider the targets when optimising its representation, even if it is used in the subsequent prediction step. For the second question, we propose a model which combines unsupervised and supervised aspects into a single architecture. The representation given by this model jointly optimises the targets and a regularisation term, which we base on CoDA-PCA [Avalos et al., 2018]. A positive answer to this question suggests an end-to-end approach may be preferable to sequentially applying dimensionality reduction and supervised learning.

Using the supervised learning algorithms of regression and classification, we investigate how the different classes of representations influence predictive performance. We focus on microbiome studies when testing these representations, for two reasons:

- The improvement in sequencing technologies have made research in this area highly active, with the potential to provide highly useful applications. The amount of data will only continue to grow as technology is further improved, magnifying the importance of research in the area.

- The data produced by these studies present several difficulties. They often have very large dimensions, as the microbial communities are large. Despite this, there are usually a relatively few number of samples per dataset. The data is also quite sparse, meaning it contains a large number of zeros. These challenges will allow us to test each approach on challenging data.

1.2.1 Contributions

Our contributions in this thesis are as follows:

- We introduce a novel end-to-end ML model for Compositional Data. This model combines supervised and unsupervised methods, optimising a representation which is a function of the targets. We show that the representations it gives can preserve features important to predicting the targets.

- We compare mathematically justified methods of Compositional Data Analysis to standard Euclidean baselines. In doing so, we find that the CoDA based methods significantly outperform their Euclidean counterparts on real Microbiome data. Furthermore, our results suggest that this performance delta increases as a function of the dimensions. This highlights the importance of proper analysis for high dimensional Compositional Data, including most microbiome studies.
- We have implemented a Python package `coda-to-end`, which provides an API for both the end-to-end model and standard CoDA methods as they are presented in this thesis.

1.3 Thesis Outline

This thesis is structured as follows:

- Chapter 2: Background. This chapter provides the reader with the necessary information to understand the design and implementation of our model. We assume the reader has some technical background, with basic linear algebra and multivariate calculus being assumed.
- Chapter 3: Design. In this chapter, we present the details of the design for our model. This includes the high level structure and specific implementation details. We also discuss the associated Python package developed as part of this project.
- Chapter 4: Experimental Methodology. This chapter presents the details of how we obtained the results in chapter 5, how to interpret them, and what data was used.
- Chapter 5: Results. Here the results of the experiments on the model are presented. We showcases the cases where the model works well, and when it does not. We discuss potential reasons why in each case.
- Chapter 6: Hyperparameter Sensitivity. This section presents additional results to complement what we show in Chapter 5. In particular, we conduct experiments which study the hyperparameters of our end-to-end model in more detail.
- Chapter 7: Conclusion. The final chapter summarises the thesis, and discusses avenues for future work in the area.

Chapter 2

Background and Related Work

In this chapter we introduce the background and motivation for this project, and related work in the area. A summary of each section is given below.

- Section 2.1: Here we introduce Compositional Data, outlining examples and areas of application. We then present the current methods used in the analysis of this data.

- Section 2.2: This section presents the topic of dimensionality reduction. We introduce the most common algorithm, PCA, as well as methods specific to compositional data. We also present several examples to illustrate the advantages of the different approaches.

- Section 2.3: The main applications of supervised learning, regression and classification, are introduced here.

2.1 Compositional Data

2.1.1 Definition

Compositional data consists of a set of non-negative vectors which sum to a constant value. The data is often normalised by setting this value to 1, allowing the components to be interpreted as percentages of a whole. Formally, a compositional matrix $\mathbf{X} \in R^{n \times d}$ will have each row vector $\mathbf{x}_i \in R^d$ as an element of the $d-1$ dimensional simplex, which is defined as:

$$S^{d-1} = \{\mathbf{s} \in R^d : \sum_{i=1}^{d} \mathbf{s}_i = c\}$$

for a fixed constant c (that is, each column vector $\mathbf{x}_i$ must sum to c). We note that the simplex is of dimension $d-1$ because we can determine the final component from the sum of the rest, given that c is a fixed constant. More precisely, the final component of a compositional vector $\mathbf{x}_i$ is clearly $(\mathbf{x}_i)_d = c - \sum_{j=1}^{d-1}(\mathbf{x}_i)_j$ and so $\mathbf{x}_i$ is of dimension $d-1$.

2.1.2 Example: RGB and the 2D simplex

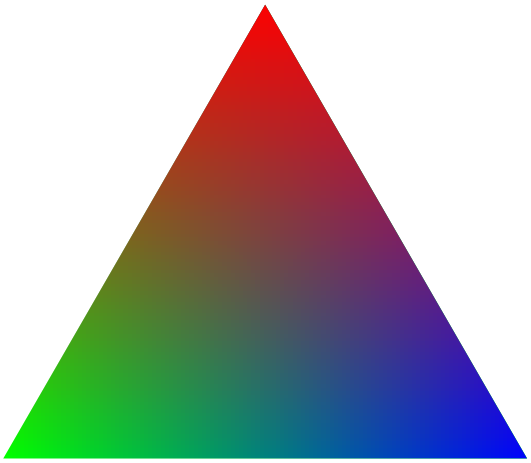
An example which illustrates this concept well is the RGB triangle. Any colour in RGB format can be interpreted as a composition of each of the colours Red, Green and Blue. In the notation above, this corresponds to a vector in R^3. Figure 2.1 gives a visual representation of the simplex of all such vectors.

Using this example, we note several points:

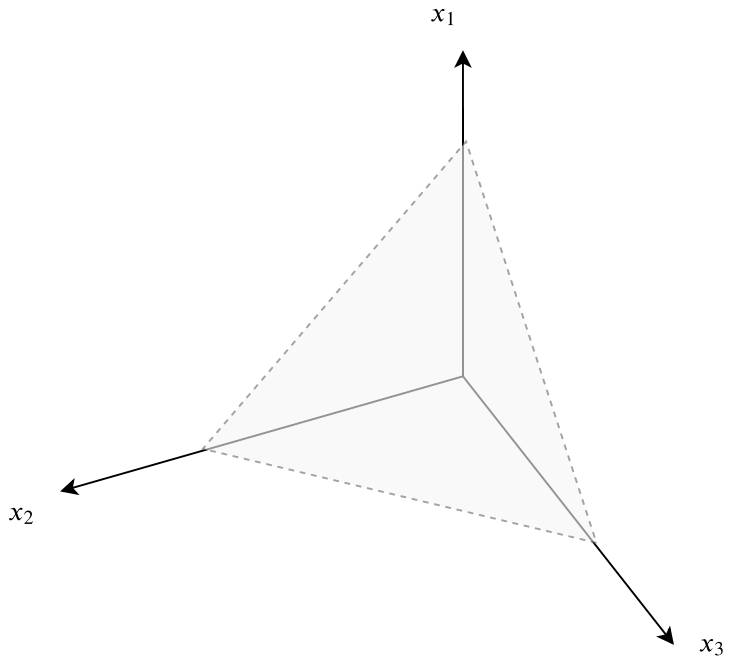
- The total magnitude of the vector (R, G, B) is irrelevant to the colour produced. What is important it the relative magnitudes between the values R, G, B. This is why we can normalise compositions to 1 without losing or distorting information.
- The simplex is fully defined as a 2D subset of R^3, as it is a triangle. In general, the simplex extends the triangle to higher dimensions.
- The maximum value any of R, G, B can take is given by each of the vertices.
- A gain in any particular R, G, B will correspond to that amount lost between the other values. This is the qualitative interpretation of the restriction that the parts must sum to a fixed value.

2.1.2.1 Subcompositions

A subset of a composition forms a composition itself. Specifically, given a composition $\mathbf{x} = (x_1, ..., x_d) \in S^{d-1}$ a subcomposition is of the form $\mathbf{z} = (z_1, ..., z_s)$, where $s < d$. We assume without loss of generality that the vectors are sorted such that $z_i = x_i$ $\forall i \in \{1, .., s\}$. Then $z_i = \frac{x_i}{\sum_{i=1}^{s} x_i}$, and so $\mathbf{z}$ is and element of the $s-1$ dimensional simplex.



(a) An RGB Colour Triangle is an example of a 2D simplex in 3D space



(b) The standard 2D simplex as a subset of R^3. The shaded region corresponds to the set $S^2 = \{\mathbf{x} \in R^3 : x_1 + x_2 + x_3 = 1\}$

Figure 2.1: The 2D Simplex

2.1.3 Compositional Data Analysis (CoDA)

Given compositional data lies on the simplex, analysis of these data are complicated by the fact that they exist in a non-Euclidean mathematical space. As such, any attempt to compare distances between elements of this space cannot take place in the standard manner. The study of Compositional Data is known as Compositional Data Analysis (CoDA for short) and Aitchison [1982] was the first to treat this issue in depth. At the core of Aitchinson's approach was the idea of a log transformation. For a point on the simplex, applying such a transformation embeds the point in Euclidean space, facilitating standard distance comparisons. There are several of these transformations, however they lead to similar results as they only differ by a linear transformation. The most widely used (and most applicable for this project) is the centered log ratio (clr) transformation. Using notation from Avalos et al. [2018], the clr transformation is formally defined as:

$$\mathbf{c}_{KL}(\mathbf{x}) = \log\left(\frac{x}{(\prod_{j=1}^{d} x_j)^{\frac{1}{d}}}\right)$$

We note that $\mathbf{c}_{KL} : S^{d-1} \to R^d$ is a bijective map from S^{d-1} to R^d by observing the inverse function which is given by

$$\mathbf{c}_{KL}^{-1}(\mathbf{x}) = e^{\mathbf{x}} \prod_{i=1}^{d} e^{\frac{x_j}{1-d}}$$

It follows that this embedding gives a unique value in R^d for all elements in the simplex S^{d-1}. The analysis of the embedded data can therefore be conducted using standard methods without any loss of information.

A common approach for practitioners who are aware of the compositional structure is to perform analysis of the log transformed data, and relate the conclusions made in the transformed space back to the original domain [Aitchison, 2003]. This approach is more mathematically grounded than using standard Euclidean methods on the simplex, but still leads to several issues. There are problems with translating interpretations from the new coordinate system back to the original domain. We cannot draw conclusions about the absolute magnitude of quantities, as compositions are inherently relative. We will soon see that the log transformation may also fail to capture relationships in the original domain (Section 2.2.3). There have been some attempts to improve upon CoDA techniques, such as hypersphere transformations [] and Dirichlet models []. As discussed by , the hypersphere approach is not fully justified mathematically. Similarly, Dirichlet models make unrealistic assumptions on the data. These issues help justify the need for further research in the area.

2.2 Dimensionality Reduction

A concept crucial to this thesis is dimensionality reduction. Broadly, these are the set of techniques used to represent high dimensional data in a reduced form. Formally, we consider a data matrix $\mathbf{X} \in R^{d \times n}$ with n elements, each with dimension d. The goal of dimensionality reduction is then to find what we denote as a representation matrix $\mathbf{A} \in R^{l \times n}$, where $l < d$. Since $l < d$, each row $\mathbf{x}_i \in R^d$ of $\mathbf{X}$ will map to a lower dimensional representation $\mathbf{a}_i \in R^l$. This can be interpreted as projection of $\mathbf{x}_i$ to a new coordinate system. In practice, it can be useful to consider $\mathbf{A}$ as an encoding of $\mathbf{X}$. We can then reconstruct an approximation to the original matrix $\mathbf{X}$ through a decoding matrix $\mathbf{V}^T \in R^{l \times d}$. Through this we can obtain a reconstruction of X of the same dimension, which is given by $V^T A \in R^{d \times n}$.

Dimensionality reduction is useful for several reasons: [Shlens, 2005]:

- The use of a low level representation of the data leads to smaller memory footprint and quicker algorithms.
- The dimensions which are removed often represent noise in the data, making it easier to find relationships between variables.
- When dimensions are highly correlated, changing the coordinate space can remove redundancy.
- Taking the low level dimension $l \leq 3$ allows for visualisation of higher dimensional data. As such, it is one of the most popular methods for exploratory data analysis.

There are many different approaches to dimensionality reduction, which seek to find $\mathbf{A}$ based on different criteria and assumptions on the underlying data $\mathbf{X}$. These approaches can be considered as minimising a distortion (generalised distance) between functions of the reconstruction $\mathbf{V^T A}$ and the original matrix $\mathbf{X}$. This interpretation was particularly helpful for the paper which inspired this project [Avalos et al., 2018]. Below we outline the specific dimensionality reduction techniques which are relevant to this research.

2.2.1 Principal Component Analysis (PCA)

The most well known dimensionality reduction method is Principal Component Analysis (PCA). There are several derivations and interpretations of PCA (e.g. Mackiewicz and Ratajczak [1993], Shlens [2005]), but they all give the same result. An intuitive explanation is given by Joliffe [2002], which we draw from to explain the concept. PCA finds the low dimensional representation $\mathbf{A}$ which explains the most variance from the original data as possible, when done through a linear transformation $\mathbf{V}$. To do this, it chooses the first basis $\mathbf{v}_1$ as the direction with maximum variance. The next basis is taken to be the direction with largest variance orthogonal to $\mathbf{v}_1$. This process continues, with $\mathbf{v}_i$ being the direction of maximum variance orthogonal to the subspace

spanned by $\mathbf{v}_1, ..., \mathbf{v}_{i-1}$. To find the $l < d$ dimensional representation of $\mathbf{X}$ we then project the data onto the first l directions of $\mathbf{V}$. Since these vectors are chosen in decreasing order, this gives us the l dimensional representation which preserves the most variance from the original data. As an example of this, Figure 2.2 illustrates the projection of a 2D Gaussian onto a single dimension. We can see here that the most information is preserved by taking the direction corresponding to the more variant axis in the original data (the x-axis in this case). We note that measurements are often not on the same scale, and if this is not addressed PCA will be biased towards larger magnitude components. It is usually necessary to standardise the data prior to PCA, ensuring 0 mean and unit variance for each component.

To show how the PCA directions are found, we will expand on the proof presented by Joliffe [2002]. From the explanation above, we know that we want the projection given by $\mathbf{v}_1$ to maximise the variance of the representation. The projection of a single point of $\mathbf{X}$ to this basis is simply $a_i = \mathbf{v}_1^T \mathbf{x}_i$, where $\mathbf{x}_i$ is a column vector of $\mathbf{X}$. The variance of the set of projected points $\{a_i\}_{i=1}^n$ is thus what we are trying to maximise. The mean of this set is

$$\bar{a} = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}_1^T \mathbf{x}_i = \mathbf{v}_1^T \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i = \mathbf{v}_1^T \bar{\mathbf{x}},$$

where $\bar{\mathbf{x}}$ is the (vector valued) sample mean. From the definition of the sample variance, we have

$$\begin{aligned}
\mathcal{V}(\{a_i\}_{i=1}^n) &= \frac{1}{n-1}\sum_{i=1}^{n}(a_i - \bar{a})^2 \\
&= \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{v}_1^T \mathbf{x}_i - \mathbf{v}_1^T \bar{\mathbf{x}})^2 \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\mathbf{v}_1^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{v}_1^T \mathbf{x}_i - \mathbf{v}_1^T \bar{\mathbf{x}}) \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\mathbf{v}_1^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i^T \mathbf{v}_1 - \bar{\mathbf{x}}^T \mathbf{v}_1) \\
&\quad \text{(since } \mathbf{v}_1^T \mathbf{x}_i \text{ are scalars, so we can rearrange the transpose)} \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\mathbf{v}_1^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i^T - \bar{\mathbf{x}}^T)\mathbf{v}_1 \\
&= \frac{1}{n-1}\sum_{i=1}^{n}\mathbf{v}_1^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\mathbf{v}_1 \\
&= \mathbf{v}_1^T\left(\frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\right)\mathbf{v}_1 \\
&= \mathbf{v}_1^T S \mathbf{v}_1. \qquad \text{(where } S \text{ is the sample variance)}
\end{aligned}$$

Note that if unconstrained, we can choose $\mathbf{v}_1$ to make the above expression arbitrarily large. As such, we impose the constraint that $\mathbf{v}_1$ must be a unit vector, such that $\mathbf{v}_1^T\mathbf{v}_1 = 1$. To solve a constrained optimisation problem, we find the stationary points of the Lagrangian [Boyd and Vandenberghe, 2009]. For our problem, we wish to maximise $\mathbf{v}_1^T S\mathbf{v}_1$ subject to the constraint that $\mathbf{v}_1^T\mathbf{v}_1 - 1 = 0$. Let us define the Lagrangian, given the multiplier $\lambda > 0$, as

$$\begin{aligned} &\mathcal{L}(\mathbf{v}_1, \lambda) = \mathbf{v}_1^T S\mathbf{v}_1 - \lambda(\mathbf{v}_1^T\mathbf{v}_1 - 1) \\ \implies &\frac{\partial \mathcal{L}(\mathbf{v}_1, \lambda)}{\mathbf{v}_1} = S\mathbf{v}_1 - \lambda\mathbf{v}_1. \end{aligned}$$

Equating this to 0 gives us the stationary point, which is where the function will be maximised. Hence the critical point is defined by

$$\begin{aligned} S\mathbf{v}_1 - \lambda\mathbf{v}_1 &= 0 \\ S\mathbf{v}_1 &= \lambda\mathbf{v}_1. \end{aligned}$$

This tells us that at our critical point, we will have that $\mathbf{v}_1$ is an eigenvector of S. Thus we need to find the eigenvector which gives the maximum variance. However, since we had the constraint that $\mathbf{v}^T\mathbf{v} = 1$, we note that

$$S\mathbf{v}_1 = \lambda\mathbf{v}_1 \implies \mathbf{v}_1^T S\mathbf{v}_1 = \mathbf{v}_1^T\lambda\mathbf{v}_1 = \lambda.$$

Thus the variance, which we are trying to maximise, is simply the eigenvalue associated with $\mathbf{v_1}$. Hence $\mathbf{v_1}$ is the eigenvector with the largest eigenvalue, which we refer to as the first principal component. To find the remaining directions, we repeat the above with the added constraint that the new direction is orthogonal to the previous principal components. We will omit this in the interest of space, but it can be shown that the i-th principal component $\mathbf{v}_i$ is the eigenvector of $\mathbf{X}$ corresponding to the i-th largest eigenvalue [Joliffe, 2002].

We note that the PCA representation $\mathbf{A} = \mathbf{V}\mathbf{X}$ expresses the original points of $\mathbf{X}$ in terms of orthogonal bases, with each coordinate being in descending order of variance. They are the same points, just in a new coordinate system. The original matrix can be fully recovered by the inverse transformation $\mathbf{V}^{-1} = \mathbf{V}^T$ since $\mathbf{V}$ is orthonormal. It is only when we are applying dimensionality reduction by taking the leading l directions of $\mathbf{V}$ that information is lost.

PCA is linked to a common process used to find orthonormal bases, the Gram-Schmidt [Leon et al., 2013] algorithm. This is used to compute the Singular Value Decomposition (SVD) of a matrix [Forsythe and Moler, 1967]. This is why the SVD can be used to find the PCA projection matrix. The SVD has been highly optimised in many current software packages, and so it is used in practice to compute the projection $\mathbf{V}$ given by PCA.

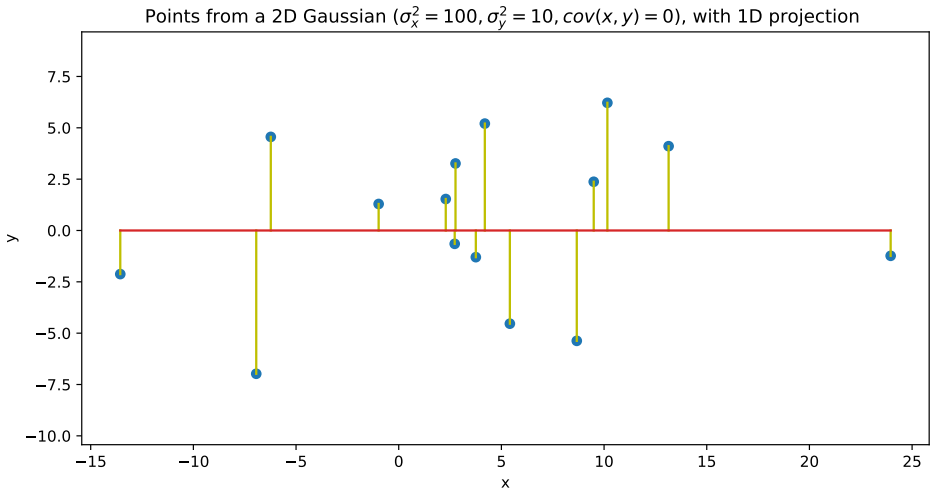


Figure 2.2: PCA projection of points from a 2 dimensional Gaussian ($\mu = 0$) to 1 dimension. The projection is constructed such that the most variance is preserved.

2.2.1.1 CLR-PCA

Standard PCA minimises the loss of the representation with respect to the original data. For the compositional case, we know this to be unjustified as the data resides on a strict subset of Euclidean space. One simple solution to this is to apply standard PCA on the log transformed coordinates introduced by Aitchison [1982]. Doing this with the CLR transformation gives us the CLR-PCA algorithm. CLR-PCA can therefore be formalised simply by replacing the original data $\mathbf{X}$ with the associated CLR transformation $\mathbf{c}_{KL}(\mathbf{X})$ in the representation loss. This gives us

$$l_{\text{CLR}-\text{PCA}} = ||\mathbf{c}_{\text{KL}}(\mathbf{X}) - \mathbf{V}^T\mathbf{A}||_F^2. \tag{2.1}$$

2.2.2 Exponential Family PCA

Compositional data often takes the form of count data. The most relevant example is microbiome data, which represent species counts in a microbial community. Count data is modelled by the Poisson distribution, which is a member of the exponential family. Exponential family PCA extends PCA from the real domain to exponential family distributions [Collins et al., 2002], and so can be used for dimensionality reduction on compositional data.

To define Exponential family PCA, we need to introduce the Bregman Divergence. This is a generalised notion of distance, comparing the similarity of 2 points. We assume a differentiable convex function $F : R^d \to R$ whose derivative is given by f.

The Bregman divergence between 2 vectors $\mathbf{x}, \mathbf{y} \in R^d$ is then given by

$$D_F(\mathbf{x}||\mathbf{y}) = F(\mathbf{x}) - F(\mathbf{y}) - (\mathbf{x} - \mathbf{y})f(\mathbf{y}).$$

Extending to matrix arguments, the Bregman divergence of matrices $\mathbf{X} = (\mathbf{x}_1^T, ..., \mathbf{x}_n^T), \mathbf{Y} = (\mathbf{y}_1^T, ..., \mathbf{y}_n^T), \in R^{d \times n}$ is

$$D_F(\mathbf{X}||\mathbf{Y}) = \sum_{i=1}^{n} D_F(\mathbf{x}_i||\mathbf{y}_i).$$

Exponential family PCA can be defined in terms of minimising a Bregman divergence, $D_{\exp}(\mathbf{V}^T\mathbf{A}||\mathbf{X})$. It is finding a representation minimising the divergence between the reconstruction $\mathbf{V}^T\mathbf{A}$ and the original data $\mathbf{X}$, for the exponential function $F(\mathbf{x}) = e^{\mathbf{x}}$.

2.2.3 CoDA-PCA

Much of this thesis builds upon the 2018 NeurIPS paper, Representation Learning for Compositional Data [Avalos et al., 2018]. The main result of this paper is the algorithm CoDA-PCA, which is a novel method for the dimensionality reduction of compositional data. The current informed approach to dimensionality reduction here is CLR-PCA, which applies standard PCA to the clr transformed data. Exponential family PCA provides an alternative, however it does not keep the data centered as with CLR [Avalos et al., 2018]. At a high level, CoDA-PCA is a combination of CLR and exponential family PCA. It keeps the data centered through CLR coordinates, and makes use of a loss function which considers the count nature of the data in the reconstruction error. The improvement can be seen dramatically in the ARMS example presented in the paper, where we see that CoDA-PCA and its variants more faithfully represent properties of the original data [Avalos et al., 2018].

2.2.3.1 Derivation Outline

To formally define CoDA-PCA, we first assume a compositional data matrix $\mathbf{X} \in R^{n \times d}$ as introduced in 2.1. We recall Exponential family PCA is given by the minimisation of the function

$$D_{\exp}(\mathbf{V}^T\mathbf{A}||\mathbf{X}).$$

To proceed, we define the convex conjugate of $F(\mathbf{x}) = \exp \mathbf{x}$ as $F^*(\mathbf{x}) = \mathbf{x} \odot \log \mathbf{x} - \mathbf{x}$, with derivative $f^*(\mathbf{x}) = \log \mathbf{x}$, and where $\odot$ denotes the element wise (Hadamard) product. We also note that the derivative of $F(\mathbf{x}) = \exp \mathbf{x}$ is $f(\mathbf{x}) = \exp \mathbf{x}$. we can apply these to the dual symmetry of Bregman divergences [Boissonnat et al., 2010],

$$D_F(\mathbf{V}^T\mathbf{X}||f^*(\mathbf{X})) = D_{F^*}(\mathbf{X}||f(\mathbf{V}^T\mathbf{A})).$$

We can now define the representation of CoDA-PCA. CoDA-PCA finds the representation for the centered log coordinates under the Exponential family PCA loss. It is

formally defined as the representation which minimises the loss

$$l_{\text{CoDA}-\text{PCA}} = D_{\exp}(\mathbf{V}^T\mathbf{A}||c_{\text{KL}}(\mathbf{X})). \tag{2.2}$$

To apply dual symmetry, we introduce the normalised matrix $\check{\mathbf{X}}$ with each observation (column vector) given by

$$\check{\mathbf{x}}_i = \frac{\mathbf{x}_i}{(\prod_{j=1}^{d}(\mathbf{x}_i)_j)^{\frac{1}{d}}}.$$

Since $f^*(\mathbf{x}) = \log(\mathbf{x})$, we note that

$$f^*(\check{\mathbf{x}}_i) = \log(\check{\mathbf{x}}_i) = \log\left(\frac{\mathbf{x}_i}{(\prod_{j=1}^{d}(\mathbf{x}_i)_j)^{\frac{1}{d}}}\right) = c_{\text{KL}}(\mathbf{x}_i.$$

And so we have that $f^*(\check{\mathbf{X}}) = c_{\text{KL}}(\mathbf{X})$. Applying dual symmetry, and recalling the conjugate of $\exp^* \mathbf{x} = \mathbf{x} \odot \log \mathbf{x} - \mathbf{x}$, we have

$$\begin{aligned}
D_{\exp}(\mathbf{V}^T\mathbf{A}||c_{\text{KL}}(\mathbf{X})) &= D_{\exp^*}(\check{\mathbf{X}}||f(\mathbf{V}^T\mathbf{A})) \\
&= D_{\exp^*}(\check{\mathbf{X}}||\exp(\mathbf{V}^T\mathbf{A})) \\
&= \sum_{i=1}^{n} D_{\exp^*}(\check{\mathbf{x}}_i||\exp(\mathbf{V}^T\mathbf{A})_i) \\
&= \sum_{i=1}^{n} D_{\exp^*}(\check{\mathbf{x}}_i||\mathbf{y}_i) \qquad \text{(setting } \exp(\mathbf{V}^T\mathbf{A}) = \mathbf{y}\text{)} \\
&= \sum_{i=1}^{n} \check{\mathbf{x}}_i \odot \log \check{\mathbf{x}}_i - \check{\mathbf{x}}_i - \mathbf{y}_i \odot \log \mathbf{y}_i + \mathbf{y}_i - (\check{\mathbf{x}}_i - \mathbf{y}_i) \odot \log \mathbf{y}_i \\
&= \sum_{i=1}^{n} \check{\mathbf{x}}_i \odot \log \check{\mathbf{x}}_i - \check{\mathbf{x}}_i + \mathbf{y}_i - \check{\mathbf{x}}_i \odot \log \mathbf{y}_i.
\end{aligned}$$

We note that this function will be minimised with respect to $\mathbf{V}^T\mathbf{A}$ and so we can omit terms which do not contain this in the CoDA-PCA loss. Therefore,

$$
\begin{aligned}
l_{\text{CoDA-PCA}} &= \sum_{i=1}^{n} \mathbf{y}_i - \check{\mathbf{x}}_i \odot \log \mathbf{y}_i && \text{(where } \mathbf{y} = \exp(\mathbf{V}^T\mathbf{A})\text{)} \\
&= \sum_{i=1}^{n} \exp(\mathbf{V^T A})_i - \check{\mathbf{x}}_i \odot \log \exp(\mathbf{V^T A})_i \\
&= \sum_{i=1}^{n} \exp(\mathbf{V^T A})_i - \check{\mathbf{x}}_i \odot (\mathbf{V}^T\mathbf{A})_i \\
&= \sum_{i=1}^{n} \exp(\mathbf{V^T A})_i - (\check{\mathbf{X}} \odot \mathbf{V}^T\mathbf{A})_i \\
&= \sum_{i=1}^{n}\sum_{j=1}^{d} \left(\exp(\mathbf{V^T A})_{ij}\right) - \sum_{i=1}^{n} (\check{\mathbf{X}} \odot \mathbf{V}^T\mathbf{A})_i \\
&= \sum_{i=1}^{n}\sum_{j=1}^{d} \left(\exp(\mathbf{V^T A})_{ij}\right) - \sum_{i=1}^{n} ((\mathbf{V}^T\mathbf{A})_i \odot \check{\mathbf{X}}) && \text{(as } A \odot B = B \odot A\text{)} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{d} \exp(\mathbf{V^T A})_{ij} - \text{trace}(\check{\mathbf{X}}^T\mathbf{V}^T\mathbf{A}).
\end{aligned}
$$

(Using the Hadamard Product identity, $\sum_i (A \odot B)_i = \text{trace}(AB^T) = \text{trace}(B^T A)$)

CoDA-PCA then finds the matrices $\mathbf{V}, \mathbf{A}$ minimising the above loss where $\mathbf{V}$ is orthonormal, and the reconstruction $\mathbf{V}^T\mathbf{A}$ is centered. We note that this loss is not convex (see 2.4), but can still converge using the bi-convex optimisation strategy outlined by Avalos et al. [2018].

2.3 Supervised Learning

2.3.1 Regression

2.3.1.1 Definition

One of the most widely used supervised learning methods is Linear Regression. The setup of regression is simply to estimate a target $y \in R$ through a linear transformation of the features $\mathbf{x} \in R^d$. For a single observation $\mathbf{x}$, this is done through the weight vector $\mathbf{w}$. The predicted value for $\mathbf{x}$ is then given by:

$$\hat{y} = \mathbf{w}^T\mathbf{x} \tag{2.3}$$

The goal is then to find $\mathbf{w}$ such that the predictions are as close as possible to the ground truth y. There is no single solution to this problem, as different formulations of distance give different results on how "close" the fit is. For example, one might prefer the properties of the l_1 distance $\sum_{i=1}^{n} |\hat{y}_i - y|$ over the l_2. The choice of distance in turn influences the optimal weights, since the weights are chosen to minimise this distance.

In most practical applications, the model weights are chosen to optimise the l_2 loss, which we define as

$$\begin{aligned} l_{\text{regression}} &= ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 && (2.4)\\ &= \sum_{i=1}^{n} ||\hat{y} - y||_2^2 && (2.5)\\ &= \sum_{i=1}^{n} ||\mathbf{w}^T\mathbf{x} - y||_2^2 && (2.6)\\ &= ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 && (2.7) \end{aligned}$$

This is often referred to as Ordinary Least Squares (OLS) regression, as it minimises the square l_2 loss. This is the loss function we assume for regression throughout the thesis.

2.3.1.2 Optimal Parameters

The optimal model parameters for the l_2 case can be found by minimising the loss in 2.7 with respect to the weights:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} l_{\text{regression}} = \arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 \tag{2.8}$$

This can easily be solved by matrix calculus:

$$\begin{aligned} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 &= (\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})\\ &= \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{y}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{w}^T\mathbf{y} - \mathbf{y}^T\mathbf{y} \end{aligned}$$

Since we are minimising with respect to $\mathbf{w}$, we can omit $\mathbf{y}^T\mathbf{y}$. Since all the terms are scalars, we may also note that $\mathbf{y}^T\mathbf{X}\mathbf{w} = (\mathbf{y}^T\mathbf{X}\mathbf{w})^T = \mathbf{X}^T\mathbf{w}^T\mathbf{y}$.

Therefore:

$$\begin{aligned} \mathbf{w}^* &= \arg\min_{\mathbf{w}} l_{\text{regression}}\\ &= \arg\min_{\mathbf{w}} \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{w}^T\mathbf{y}\\ \nabla_{\mathbf{w}} l_{\text{regression}} &= 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} \end{aligned}$$

Setting this to 0 gives us:

$$\begin{aligned} 2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} &= 0\\ \mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{y}\\ \mathbf{w} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

This solution is a minimum of the loss function $l_{\text{regression}}$ since the Hessian $2\mathbf{X}^T\mathbf{X}$ is positive definite for all $\mathbf{X}$. Therefore the optimal set of weights for the l_2 regression

loss is given by

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{2.9}$$

2.3.1.3 Principal Component Regression

A common approach in regression problems is to apply dimensionality reduction on the features, and use the resulting representation as input for the regression. This is particularly useful for large feature spaces, and can improve model performance and efficiency. We expand on why this occurs in 2.3.4. This approach is known as Principal Component Regression (PCR) [Liu et al., 2003]. There are several drawbacks to this sequential application of algorithms. The main problem is that the representation is not chosen with respect to the targets. It is simply applying PCA. The goal of PCR is to optimise the regression loss, and there is no a priori reason to assume that the targets will be functions of the most variant axes. In comparison to standard regression, there is less interpretability in the model, since the targets are now being fit to a new coordinate space.

2.3.2 Classification

2.3.2.1 Definition

The other key supervised learning problem is classification. In a classification task, the targets $\mathbf{y}$ are now in a discrete set $\{1, ..., k\}$ which we call the classes. Given an observation $\mathbf{x} \in R^d$, the goal is then to predict which class it belongs to i.e. learn a mapping $\mathbf{x} \to \mathbf{y} \in \{1, ..., k\}$. This is referred to as a multi class classification problem, in contrast to the binary case where $\mathbf{y} \in \{0, 1\}$. There are myriad ways to solve this problem in the literature. We focus on the multi class logistic regression approach, as the associated loss function will be needed for our model.

2.3.2.2 Multi class Logistic Regression

Multi class Logistic Regression approaches the classification problem through combining a softmax function with a linear transformation of the features. Formally, for a vector $\mathbf{x} \in R^d$ we define softmax as a function on each component $x_i \in \mathbf{x}$:

$$S(x_i) = \frac{e^{x_i}}{\sum_{x_i \in \mathbf{x}} e^{x_i}} \tag{2.10}$$

As with Regression, we have a linear transformation of the features $\mathbf{w}^T\mathbf{x}$. The softmax is then applied to this transformation, which gives the predicted probabilities for each class:

$$P(\hat{y} = i|\mathbf{w}^T\mathbf{x}) = S((\mathbf{w}^T\mathbf{x})_i) \tag{2.11}$$
$$= \frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}} \tag{2.12}$$
$$:= p_i \tag{2.13}$$

The prediction is then taken as the maximum of these probabilities:

$$\hat{y} = \arg\max_i S((\mathbf{w}^T\mathbf{x})_i) \tag{2.14}$$
$$= \arg\max_i \frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}} \tag{2.15}$$
$$= \arg\max_i p_i \tag{2.16}$$

2.3.2.3 Negative Log Likelihood

To learn the weights $\mathbf{w}$, we again must optimise a loss function. The loss function which we use is the Negative Log Likelihood. We justify the use of this loss function by considering the likelihood expression for $\mathbf{y}$. The likelihood denotes the probability of observing a particular set of targets under a model given the parameters and features. For a given set of targets $\mathbf{y}$, we specify the outcome of the jth component y_j using a one hot encoding. This encoding assigns to each y_j a vector in $\{0,1\}^k$. For $i \in \{1,..,k\}$, each component $(k_j)_i \in \{0,1\}$ of y_j acts as a binary indicator for whether y_j is in class i. For example, if y_j is in class 2, then under the one hot encoding $y_j = (0,1,0,0,..)$. Under this encoding, the likelihood of a single observed target y_j given the corresponding feature vector $\mathbf{x_j}$ and parameters $\mathbf{w}$ is therefore:

$$P(y_j|\mathbf{w}^T\mathbf{x}_j) = \prod_{i=1}^{k} P(\hat{y} = i|\mathbf{w}^T\mathbf{x}_j)^{(k_j)_i} \tag{2.17}$$
$$= \prod_{i=1}^{k} \left(\frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}} \right)^{(k_j)_i} \tag{2.18}$$
$$= \prod_{i=1}^{k} p_i^{(k_j)_i} \tag{2.19}$$
$$= p_i^{j'} \qquad \text{(where } j' \text{ is the index where } (k_j)_i = 1\text{)} \tag{2.20}$$

Put simply, the likelihood of an observed y_j is just the probability under the model of the class which y_j is observed to be. The likelihood of the whole set of targets $\mathbf{y}$ is thus given by:

$$\mathcal{L}(\mathbf{y}) := P(\mathbf{y}|\mathbf{w}, \mathbf{X}) \tag{2.21}$$

$$= \prod_{j=1}^{n} P(y_j|\mathbf{w}^T\mathbf{x}_j) \qquad \text{(Since the observations are independent)}$$

$$= \prod_{j=1}^{n}\prod_{i=1}^{k}\left(\frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}}\right)^{(k_j)_i} \tag{2.22}$$

$$= \prod_{j=1}^{n}\prod_{i=1}^{k} p_i^{(k_j)_i} \tag{2.23}$$

$$\tag{2.24}$$

To find the optimal weights, the maximum likelihood perspective is to take the weights in such a way as to maximise the above likelihood. Now we note that since log is a monotonically increasing function, the maximum x value of the log likelihood is the same as the maximum of the original likelihood. Maximising log likelihood is a common approach, since it often removes exponents and makes for simpler calculations. From our perspective, we want to find a loss function to minimise. We can express the maximisation of the log likelihood as the minimisation of the negative log likelihood. Hence through minimising the negative log likelihood, we will find the weights which give the maximum likelihood solution. We write this explicitly as:

$$-\log \mathcal{L}(\mathbf{y}) = -\log \prod_{j=1}^{n}\prod_{i=1}^{k} p_i^{(k_j)_i} \tag{2.25}$$

$$= -\sum_{j=1}^{n}\sum_{i=1}^{k} \log(p_i^{(k_j)_i}) \tag{2.26}$$

$$= -\sum_{j=1}^{n}\sum_{i=1}^{k} (k_j)_i \log p_i \qquad \text{(where } p_i = \frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}}\text{)}$$

We thus have the above expression forming the loss function to optimise. Unlike the regression loss, the above loss function has no know analytical solution **??**. It must therefore be optimised using gradient based methods discussed in the following section, 2.4.

2.3.3 Neural Networks

2.3.3.1 Definition

Artificial Neural networks form the basis for most modern machine learning models. They are comprised of computational units referred to as neurons. These neurons are stacked into separate layers, with the input for each neuron being given by the output of all previous layers. Formally, we specify a neural network with k layers by

the set of layers $\{\mathbf{x}_i\}_{i=1}^{k}$. Each layer $\mathbf{x}_i$ has $j(i)$ neurons, which we define as $\mathbf{x}_i = ((x_i)_1, ..., (x_i)_{j(i)})$. With the exception of the input layer, the input of each neuron is given by the output of all neurons in the previous layer. This is referred to as a fully connected network, since every neuron is connected to all neurons in the preceding layer. There are many alternative setups, but we focus on this case as it is the only one of relevance to our work. The inputs for each neuron from the previous layer are weighted, and it is these weights which the model optimises. We denote the weight connecting neuron j in layer i with neuron z in layer $i+1$ as $w_{jz}^{(i)}$. For notational ease, we also add a neuron $(x_i)_0$ for each layer with a constant output of 1. The corresponding weight $w_{0j}^{(i)}$ forms the bias term for neuron j in layer $i+1$. The output for the z-th non-bias neuron in layer $i+1$ is then defined by

$$(x_{i+1})_z = h\left(\sum_{j'=0}^{j(i)} w_{j'z}^{(i)}(x_i)_{j'}\right). \tag{2.27}$$

We say that h is an activation function, and it is assumed to be non-linear and differentiable. We can express these outputs in vector format, and in doing so give a concise expression for the whole network. We may rewrite 2.27 as

$$(x_{i+1})_z = h\left((\mathbf{w}_z^{(i)})^T \mathbf{x}_i\right),$$

where we define $\mathbf{w}_z^{(i)}$ as the set of weights from layer i to neuron z in layer $i+1$. With a slight abuse of notation setting h to be vector valued, we can specify the output for the whole layer $\mathbf{x}_{i+1}$ by

$$\mathbf{x}_{i+1} = h\left(\mathbf{W}_i^T \mathbf{x}_i\right) := \left(h\left((\mathbf{w}_1^{(i)})^T \mathbf{x}_i\right), ..., h\left((\mathbf{w}_{j(i+1)}^{(i)})^T \mathbf{x}_i\right)\right),$$

where $\mathbf{W}_i = \left(\mathbf{w}_1^{(i)}, ..., \mathbf{w}_{j(i+1)}^{(i)}\right)$. Hence the output layer will be given by

$$\begin{aligned}\mathbf{x}_k &= h\left(\mathbf{W}_{k-1}^T \mathbf{x}_{k-1}\right) = h\left(\mathbf{W}_{k-1}^T \left(h\left(\mathbf{W}_{k-2}^T \mathbf{x}_{k-2}\right)\right)\right) \\ &= h\left(\mathbf{W}_{k-1}^T \left(h\left(\mathbf{W}_{k-2}^T \left(h\left(\cdots\cdots h\left(\mathbf{W}_1^T \left(h\left(\mathbf{W}_0^T \mathbf{x}_0\right)\right)\right)\right)\right)\right)\right)\right),\end{aligned}$$

where the input vector is $\mathbf{x}_0$.

2.3.3.2 Universal Approximation

The setup of neural networks described above allows for a very broad range of applications. This is formalised by the Universal Approximation Theorem [Csáji, 2001], which states that any real valued continuous function can be approximated by a network as described above. This extends to vector valued functions (see for example, [Zaresky-Williams, 2019]). As the inputs and outputs are of arbitrary size, we can therefore use neural networks to solve supervised learning problems with arbitrary labels and targets. Provided we have a differentiable loss function to capture the task, we can apply the optimisation procedure outlined in 2.4 to find weights of the network optimised for

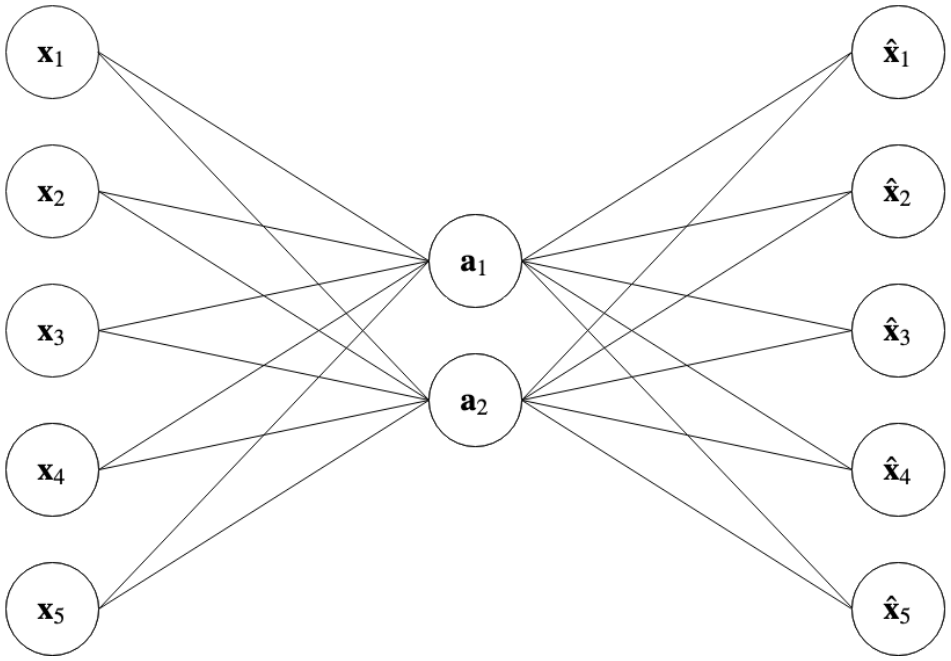


Figure 2.3: Example Autoencoder network with input dimension $d = 5$ and latent (representation) dimension $l = 2$. Notice the 'bowtie' shape, with a small inner layer connecting the symmetric input and output layers.

the problem.

2.3.3.3 Autoendcoders

The main application of neural networks that we are concerned with is the autoencoder [Kramer, 1991]. In its simplest form (and how it is described by Kramer [1991]), an autoencoder is simply a fully connected network with:

- A final layer of the same dimension d as the input.
- A bottleneck layer with $l < d$ neurons.
- An l_2 loss between the input and output neurons.

Figure 2.3 depicts an example autoencoder network with an input dimension of $d = 5$ and latent dimension of $l = 2$. The input matrix $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_d)$ is approximated by the reconstruction $\hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_d)$. By setting the loss as the squared error between $\mathbf{X}$ and $\hat{\mathbf{X}}$, optimising the network leads to a representation $\mathbf{A} = (\mathbf{a}_1, ..., \mathbf{a}_l)$ which captures as much information (judged by the squared loss) about $\mathbf{X}$ as possible.

2.3.4 Reconstruction Error

It is widely known that the use of dimensionality reduction prior to the application of supervised learning models can improve performance (see, for example, Howley et al.

[2006]). There is some speculation as to exactly why this is the case, as these methods reduce the amount of information present in the data. The improvement in accuracy is mostly seen for high dimensional data, for which the widely known "curse of dimensionality" reduces model performance. The reduction of the input dimension facilitates lower computational cost and faster convergence of algorithms, and so greater performance. In practical cases of small training examples, higher dimensional data prevents many models from recognising patterns in noisy data. It is also claimed that principal components explaining less variance correspond to noise in the data. Under this view, the removal of these components would help to reduce overfitting by ensuring the model is fitted only to the useful parts of the data. Whatever the reason, supervised learning performance can be improved by dimensionality reduction on the features, and this forms the motivation for our model. Conventionally, dimensionality reduction is first performed to obtain a low level representation of $\mathbf{X}$, which we denote as A. The matrix $\mathbf{A}$ is then used as input to a supervised learning algorithm. Such an approach constructs $\mathbf{A}$ with no regard to the performance of the supervised learning, only optimising based on a reconstruction loss.

2.4 Optimisation

Most machine learning problems can fundamentally be considered as optimisation tasks, which focus on finding the extrema of functions under potential constraints. For example, OLS regression is an optimisation problem as it is minimising the l_2 loss. These functions can occasionally be solved analytically, like in the regression case. It is much more common to use gradient based optimisation algorithms, as the analytic solutions are often intractable or nonexistent. Many variants of these algorithms exist, but they are all based on the idea of gradient descent. Given a differentiable function $f(\mathbf{x}) : R^d \rightarrow R$, the gradient descent algorithm proceeds by iteratively taking a step down in the direction of the gradient $\nabla_{\mathbf{x}} f(\mathbf{x})$. We first specify initial value $\mathbf{x}_0$, and a learning rate η to control the rate of descent. We can then define the update step as:

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \eta \nabla_{\mathbf{x}} f(\mathbf{x}) \tag{2.28}$$

In certain cases, the objective (i.e. the function to be optimised) is convex and so guaranteed to have a unique minimum [1]. In these cases, the gradient descent algorithm will converge to the minimum value. Many problems are non-convex and so optimisation through gradient based methods is not guaranteed to reach a global minima. In these cases, the algorithm may converge to a local minimum, or reach an inflection (saddle) point.

Understanding this is an important point, as it can help explain why models do not converge. To illustrate this, we consider the following functions:

[1] This does not guarantee the existence of the minimum, as we saw with the negative log likelihood loss used for logistic regression

$$z = x^2 + y^2 \qquad \text{(convex)}$$
$$z = x^3 - 10x + 2y^2 \qquad \text{(non-convex)}$$

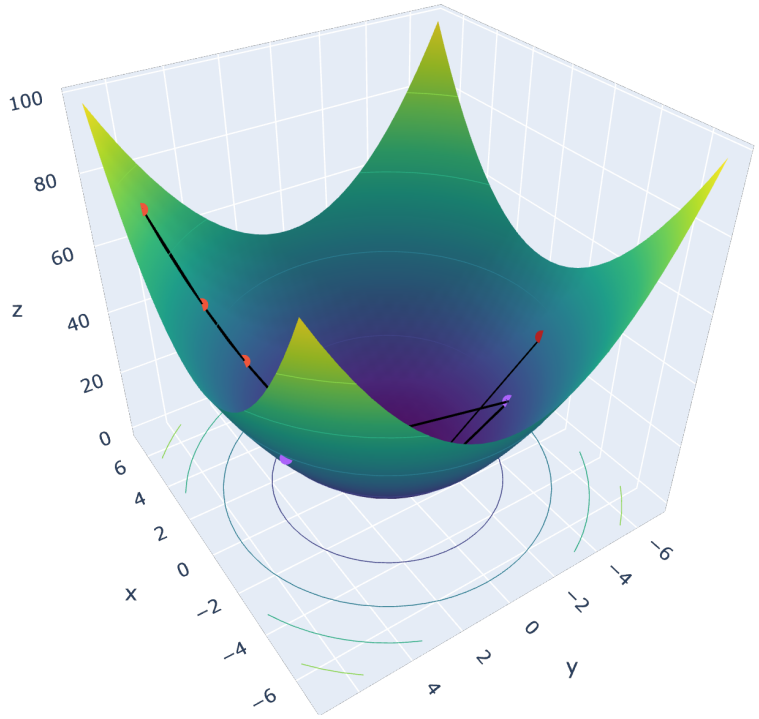
Figures 2.4 and 2.5 provide 3D heatmaps of these functions, where darker colours correspond to a smaller value. The function values following 10 iterations of gradient descent are also shown. The different colours correspond to different initial values. For the convex function in 2.4, all initialisations lead to the same point. This is the global minimum of the function, which can easily be seen between the two figures. The function in 2.5 is unbounded below, and hence has no minimum. We can see that for all 3 initialisations there was a different result:

- The red (top) initialisation is approaching a local minima. Once it has reached this it has nowhere to go, since the gradient is 0 at this point.
- The brown (middle) initialisation is stuck at a saddle point, as there are neighbouring points with a lower value. Since the gradient is 0, the algorithm will not move to find them.
- The purple (bottom) initialisation is descending the function, and will continue to do so as it is unbounded below

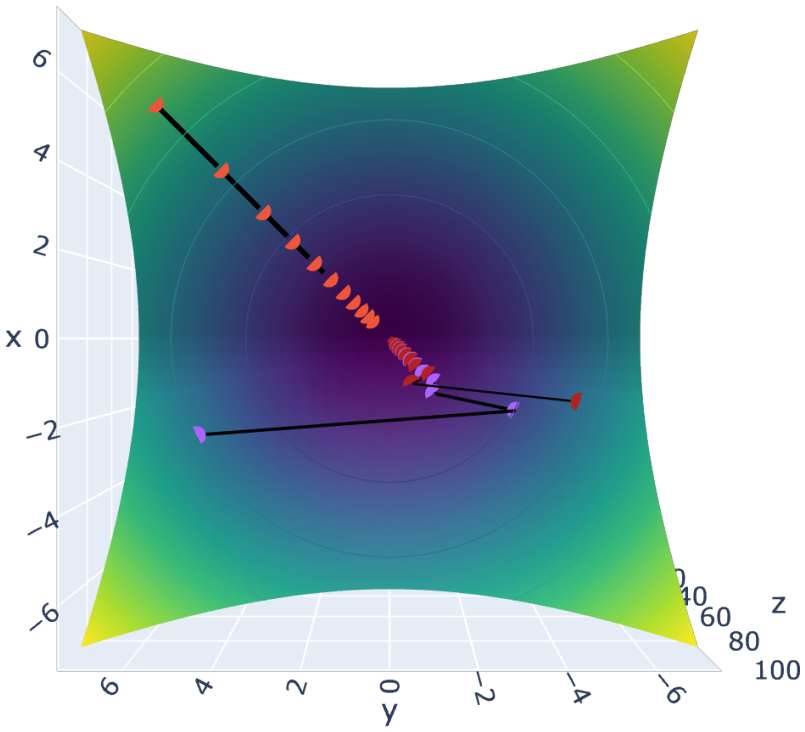
Solutions to the problems exhibited in Figure 2.5 are usually ad hoc, with there rarely being theoretical guarantees on convergence. In practice however, they do improve the chances of model convergence. Examples include variable learning rates, where the learning rate decreases (or increases) over time. This can cause the optimisation algorithm to do things like speed up convergence, slow it down or "jump" out of minima. As is pictured in Figure 2.5, random initialisations are another strategy, with some initialisations doing well (others not so much). There are many algorithms to approach the problem of non-convex optimisation. The most popular, and arguably one of the best is the Adam (derived from Adaptive moment) [Kingma and Ba, 2014] optimiser. Without going in to too much detail, this algorithm uses variable learning rates which are distinct for each parameter, among a host of other improvements.

2.5 Microbiome

One of the main fields of interest for Compositional Data Analysis is microbiome studies. At a high level, a microbiome refers to the community of simple organisms (e.g. bacteria) which occupy a particular location on any complex organism (e.g. humans). This is a rapidly growing subfield of genomics, as it can reveal relationships which exist between the microbiome and the external environment. One of the main applications of this includes studying the human microbiome to investigate its connection to health and disease, with the American Gut Project [McDonald et al., 2018] making recent progress in this area.

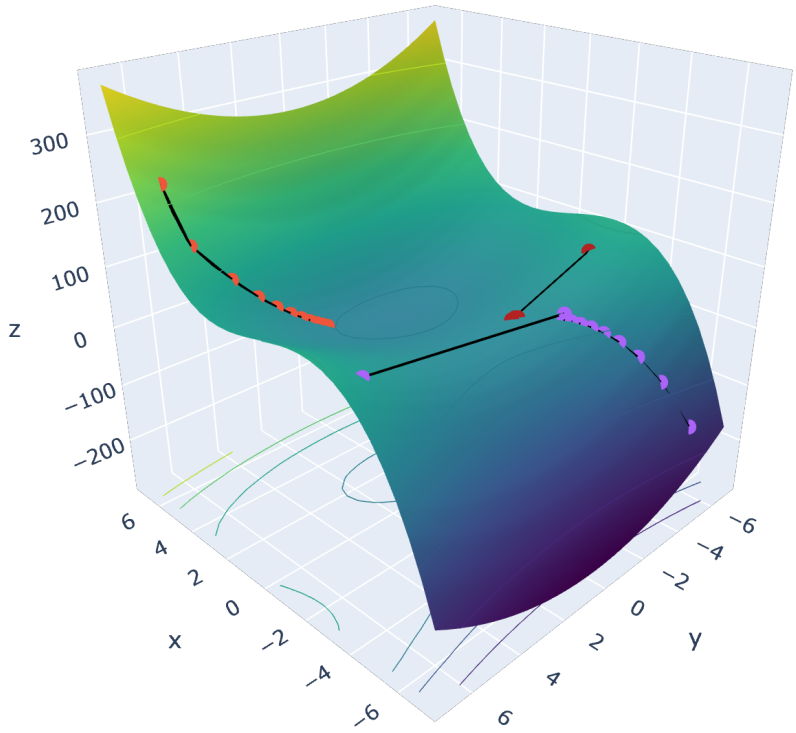


(a) 3D surface of the convex function $z = x^2 + y^2$

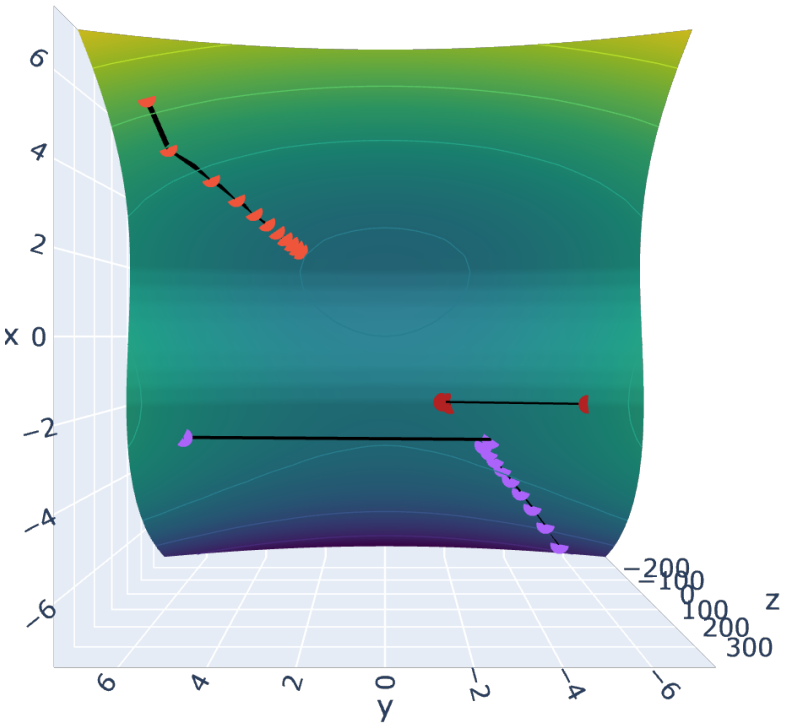


(b) Overhead perspective of the function in (a)

Figure 2.4: Gradient Descent on a convex function given different initialisations. All initialisations converge to the same value, which is the global minimum



(a) 3D Surface of the non-convex function $z = x^3 - 10x + 2y^2$



(b) Overhead perspective of the function in (a)

Figure 2.5: Gradient Descent on a non-convex function given different initialisations. All initialisations lead to different values; top is stuck in a local minima, middle is stuck at a saddle point where $\nabla z = 0$, bottom will keep descending as z is unbounded

2.5.1 Genomics and Gene Sequencing

Before we can understand how CoDA is relevant for Microbiome studies, we must cover what exactly the data is and how it is collected. As mentioned above, microbiome data is collected to reveal information about the distribution of smaller organisms in a particular site on another larger organism (the human gut microbiome, for example). The goal is to fully describe this microbial community by the abundance of each organism inside it. The first step is to sequence the community of interest. There are two broad approaches to this, which are discussed in 2.5.1.1 and 2.5.1.2 [Li, 2015]: Marker gene sequencing (usually 16S sequencing), and metagenomic 'shotgun' sequencing. Once this is completed, the next step is to use the resulting sequences to 'bin' them to the organism they belong to. The process of classifying organisms from reads is referred to as taxonomic profiling, with the identified organisms sometimes being referred to as 'taxa'. This process is dependent on the sequencing method used, and in practice it is difficult to map from sequences to frequencies. One problem is how the sequences are mapped to individual organisms. This is again highly dependent on the sequencing method, which we detail in the following sections.

2.5.1.1 16S Sequencing

Although many species have been identified to date, there are a vast number of microorganisms which do not fit into the current taxonomy. This poses a large problem, as such organisms cannot be classified by reference to those that are currently known. One solution to the problem is to use Operational Taxonomic Units, or OTUs. This relates to one of the common gene sequencing methods, 16S sequencing [Nguyen et al., 2016]. In this sequencing paradigm, only a single gene (the 'marker' gene) is sequenced for a given sample, which is usually the 16S rRNA gene. This specific gene is used since it is common to all bacteria, and its function has changed minimally over time. Variations in the 16S gene can therefore be linked to the evolution of an organism. Using a reference database mapping the marker genes of known species, we can profile the organisms. For example, a similarity of below 97% between 16S sequences is widely considered to distinguish organisms at the species level [Janda and Abbott, 2007]. These similarities are then used to construct OTUs, grouping organisms into a single OTU if they fall within a threshold value (e.g. 97% for species). From this, an OTU table is constructed which contains counts of each OTU per sample. This data is intrinsically compositional, since the total read count of the instrument is fixed, and so the OTU counts are relative proportions [Gloor et al., 2017].

2.5.1.2 Metagenomic Sequencing

The other sequencing paradigm we are interested in is metagenomic sequencing. Marker gene methods such as 16S sequence a single gene across organisms, whereas metagenomic sequencing maps the genome of each organism [Bragg L, 2014]. This allows for a more detailed representation of the underlying microbiome community. Hence the use of the term "meta"genomics, as it maps all the genomes (which represent the entirety of

DNA found in an organism) from the different organisms in a sample. Assembly of the genomes involves piecing together the whole genome from sequences of reads of base pairs. These reads resemble random draws from the "distribution" of genes given by the total sample[Bragg L, 2014] . Hence it is often referred to as shotgun sequencing, with the distribution of reads resembling the spread pattern of a shotgun. Since the read fragments are not necessarily linked, metagenomic sequencing is prone to errors. It also requires that reads are organised and mapped to genomes of particular organisms. Taxonomic profiling of these organisms is again done by reference to an existing database [Bragg L, 2014].

2.6 Summary

In this chapter we have presented the background required to understand the design and experiments of our model. We covered four main topics:

- We formally defined Compositional Data as data on the simplex, through the example of the RGB triangle. We then briefly summarised the current approaches to Compositional Data Analysis and its limitations
- We presented the concept of dimensionality reduction, discussing the high level approach and its associated advantages. We introduced PCA, CLR-PCA and CoDA-PCA as the algorithms which we will need for our model.
- The two supervised learning methods we use in the next chapter were presented. These being OLS Regression and multi class classification through Logistic Regression. We also discuss the limitations with current approaches for combining dimensionality reduction and supervised learning, as this motivates the architecture of our model. We briefly discuss Neural Networks and Autoencoders since they are used to construct our model. We present Machine Learning as the optimisation of functions. The associated gradient descent algorithm and its limitations were expanded upon here, as a variant of this is used to optimise the model in 3.1.
- Finally, we present an overview of Microbiome studies. We discuss how the data are collected, processed, and why they are compositional.

In the following chapter, we present the design of our model for the end to end learning of compositional data.

Chapter 3

Design and Implementation

In this chapter we introduce a novel model for Compositional Data Analysis. It combines dimensionality reduction and supervised learning into a single end-to-end algorithm. This design is motivated by the drawbacks of traditional methods, such as PCR. Such methods obtain the low level representation independently of the training targets, and so may ignore information which can improve the predictive performance. We divide this chapter into the following sections:

- Section 3.1 presents a high level overview of the model, including the assmuptions, inputs and outputs.
- Section 3.2 describes in detail the loss function(s) which the model is optimising.
- Section 3.3. The full architecture of the model is outlined here. We also detail the computations involved in a forward pass of the model.
- Section 3.4. The implementation details of the model are presented and discussed here. We also detail the Python package for this code, which is available online.

3.1 Model Overview

The application of our model is to supervised learning problems where the features form a compositional data set. The targets are currently restricted to single real valued numbers (regression), or classification problems (either single or multi class). Formally, the model takes as input a compositional matrix $\mathbf{X} \in R^{n \times d}$, with n samples each of dimension d. Each sample $\mathbf{x}_i$ corresponds to the composition $(x_1, ..., x_d) \in S^{d-1}$, and forms the i-th row of $\mathbf{X}$. The output of the model is a prediction $\hat{\mathbf{y}}$ of the true targets $\mathbf{y}$. In the regression case, $\mathbf{y} \in R^n$, and for classification $\mathbf{y} \in \{0, ..., k\}^n$ where k is the number of classes. The prediction is generated from the following two steps:

- Reduce $\mathbf{X}$ to a representation $\mathbf{A} \in R^{n \times l}$ where $l < d$, through the mapping $\Theta(\mathbf{X}) = \mathbf{A}$.
- Use $\mathbf{A}$ as input to the prediction mapping, which we denote as $r(\mathbf{A}) = \hat{\mathbf{y}}$.

We can then define the end-to-end mapping from $\mathbf{X}$ to $\hat{\mathbf{y}}$ as the composition of the above functions:

$$f(\mathbf{X}) = r(\Theta(\mathbf{X})) = r(\mathbf{A}) = \hat{\mathbf{y}} \tag{3.1}$$

3.2 End to End Loss

The steps outlined above for the forward pass are common to many supervised learning approaches, such as PCR which also performs regression on a low level representation. The novelty of our approach is the loss function under which $f(\mathbf{X})$ is optimised. In typical applications, the encoding $\Theta(\mathbf{X})$ and the prediction $r(\mathbf{A})$ are separately optimised with respect to different criteria. our model instead directly optimises the composition $f(\mathbf{X})$ under a single loss function.

To define this loss function we first recall the regression and classification loss from 2.3, and the CoDA-PCA loss from 2.2.3. For predictions $\hat{\mathbf{y}}$, targets $\mathbf{y}$, reconstruction $\mathbf{Y} = \mathbf{V}^T\mathbf{A}$ and geometrically normalised data $\check{\mathbf{X}}$ we have:

$$l_{\text{regression}} = ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 \tag{3.2}$$

$$l_{\text{classification}} = -\sum_{j=1}^{n}\sum_{i=1}^{k}(k_j)_i \log p_i \qquad (\text{where } p_i = \frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}})$$

$$l_{\text{CoDA-PCA}} = \mathbf{1}^T \exp(\mathbf{Y})\mathbf{1} + \text{tr}(\check{\mathbf{X}}^T\mathbf{Y}) \tag{3.3}$$

We have claimed that our model provides an end to end solution for dimensionality reduction and supervised learning. This is done is by combining the reconstruction loss $l_{\text{CoDA-PCA}}$ with a supervised loss, $l_{\text{regression}}$ or $l_{\text{classification}}$. This combination is taken as a weighted sum of the reconstruction and supervised losses, with a scaling parameter λ. We note that all the above loss functions are differentiable, and so the weighted sum is differentiable.

Formally, given a reconstruction loss l_x, supervised loss l_y, and a scalar $\lambda \in R$ we define the end-to-end loss as:

$$l_{\text{end}-\text{to}-\text{end}} = l_y + \lambda l_x \tag{3.4}$$

We note that this concept can be applied to any choice of reconstruction or supervised loss l_x or l_y, however we focus on the CoDA case with single valued regression and multi class classification.

If we only optimise with respect to the supervised loss l_y, it may lead to representations $\mathbf{A}$ which do not faithfully represent the data (We confirm this experimentally in 6.1). An inaccurate representation can reduce predictive performance when generalising beyond the training data. This motivates the addition of the reconstruction loss l_x, which in many ways is similar to a regularisation term. This penalises the model if its encoding deviates too far from an accurate representation. Such an approach prevents pathological representations which are overly optimised to the training data, and so should improve performance when scaled appropriately. We use the λ term to control this scaling, which allows us to tune the balance between the losses. We can now proceed to define the specific loss functions for each case.

3.2.1 CoDA-Regress

In the regression case, we are given targets and predictions $\mathbf{y}, \hat{\mathbf{y}} \in R^n$. We define the algorithm CoDA − Regress using the loss

$$l_{\text{CoDA}-\text{Regress}} = ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 + \lambda \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \exp \mathbf{Y}_{ij} + \text{tr}(\check{\mathbf{X}}^T \mathbf{Y}) \right), \tag{3.5}$$

Where $\mathbf{Y}$ is the reconstruction of $\mathbf{X}$ given by the forward pass.

3.2.2 CoDA-Cl

For a classification problem, we have targets and predictions $\hat{\mathbf{y}}, \mathbf{y} \in \{1, ..., k\}$. We define the loss function for the algorithm CoDA − Cl as

$$l_{\text{CoDA}-\text{Cl}} = -\sum_{j=1}^{n} \sum_{i=1}^{k} (k_j)_i \log p_i + \lambda \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \exp \mathbf{Y}_{ij} + \text{tr}(\check{\mathbf{X}}^T \mathbf{Y}) \right),$$
$$(\text{where } p_i = \frac{e^{(\mathbf{w}^T \mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T \mathbf{x})_{k'}}})$$

Where $(k_j)_i$ denotes the value of $\mathbf{y}_j$ for the i-th class under the one hot encoding.

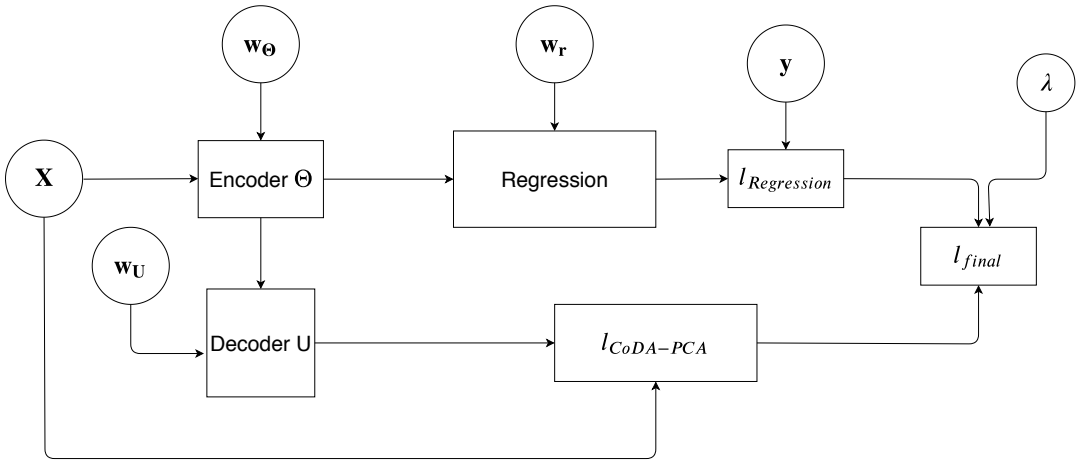


Figure 3.1: End to End ("Coda-to-end") Model Diagram

3.3 Model Architecture

We are now in a position to fully describe the architecture of the model. Recall the forward pass from 3.1:

$$f(\mathbf{X}) = r(\Theta(\mathbf{X})) = r(\mathbf{A}) = \hat{\mathbf{y}}$$

Both Θ and r are implemented as fully connected networks, to which we associate the weights $\mathbf{w}_\Theta$ and $\mathbf{w}_r$. We additionally define a decoder network U with weights $\mathbf{w}_U$. This network takes as input the representation $\Theta(\mathbf{X}) = \mathbf{A}$, and outputs a reconstruction $\mathbf{Y}$ of $\mathbf{X}$ in the original domain. This autoencoder structure was inspired by the CoDA-AE algorithm in Avalos et al. [2018]. Although not necessary for the forward pass, the reconstruction is needed for optimisation of the model weights. These model weights are given by the union of the above networks:

$$\mathbf{W}_f = \{\mathbf{w}_\Theta\} \cup \{\mathbf{w}_r\} \cup \{\mathbf{w}_U\} \tag{3.6}$$

These weights are optimised with respect to the end-to-end loss functions defined in 3.2. Given these functions are differentiable, they can be optimised by any gradient based optimisation algorithm. Algorithm 1 details the high level training cycle of the model. Figure 3.1 outlines the information flow for computing the combined loss.

Algorithm 1: coda-to-end: An end-to-end model for CoDA

input: Compositional data matrix $\mathbf{X}$, initialisation matrix I, ground truth $\mathbf{y}$, scalar λ, optimiser O, Encoder network Θ, Decoder network U, prediction layer r, epochs n

output: Trained model weights $\mathbf{W}_f$

$\mathbf{W}_f = \mathbf{I}$;

$epoch = 0$;

while $epoch < n$ **do**

  $\mathbf{A} = \Theta(\mathbf{X})$;

  $\hat{\mathbf{y}} = r(\mathbf{A})$;

  $\mathbf{Y} = U(\mathbf{A})$;

  $l_{\text{CoDA}-\text{PCA}} = \mathbf{1}^T \exp(\mathbf{Y})\mathbf{1} + \text{tr}(\check{\mathbf{X}}^T\mathbf{Y})$;

  $l_x = l_{\text{CoDA}-\text{PCA}}$;

  if $\mathbf{y} \in R^n$ **then**

    $l_y = ||\hat{\mathbf{y}} - \mathbf{y}||_2^2$;

  else

    $p_i = \frac{e^{(\mathbf{w}^T\mathbf{x})_i}}{\sum_{k'=1}^{k} e^{(\mathbf{w}^T\mathbf{x})_{k'}}}$;

    $l_y = -\sum_{j=1}^{n}\sum_{i=1}^{k}(k_j)_i \log p_i$, $((k_j)_i = 1$ if $\mathbf{y}_j$ is in class i, 0 otherwise) ;

  end

  $l_{coda-to-end} = l_y + \lambda l_x$;

  $\mathbf{W}_f = O(\mathbf{W}_f, l_{coda-to-end})$;

  $epoch = epoch + 1$;

end

3.4 Implementation

The model defined in 3.3 was implemented in PyTorch, with the source code available on GitHub[1]. A high level outline of the implementation is as follows: The algorithms CoDA-Regress and CoDA-Cl form the basis for the code, and are each implemented as a separate class. The user defines the desired low level dimension, as well as the encoder and decoder dimensions. Using these parameters, an ordered dictionary is used to stack the layers for the encoder and decoder to ensure the correct ordering of dimensions. This includes the nonlinearities, for which we use Exponential Linear Units (ELUs). These dictionaries are then used as input to the PyTorch Sequential module to format them as PyTorch objects.

The output layer is algorithm specific. CoDA-Regress has a linear layer mapping

[1] https://github.com/sean-lamont/coda-to-end

to a single output, which is the predicted regression value. CoDA-Cl has a linear layer mapping to the number of classes followed by a log softmax layer, which gives the log probabilities for each class.

The forward pass is as described in 3.3, with the input being modified to contain both the original matrix X and $X_{ckl} = \log \check{X}$. This is done as per Avalos et al. [2018].

The optimisation is done using Adam [Kingma and Ba, 2014]. Referring to algorithm 1, this takes the place of the function O. We found this to be more numerically stable during experiments in comparison to standard Stochastic Gradient Descent (SGD), in addition to faster convergence.

3.4.1 CoDA-PCA Package

We see from [Avalos et al., 2018], CoDA-PCA gives a reconstruction which can be more representative of the original structure than other methods. Although the code for this paper is available, it has not yet been packaged to production standards. We have reimplemented and packaged the code for CoDA-PCA, along with the CoDA-Regress and CoDA-Cl algorithms as they are presented here. These are available on the Python Packaging Index (PyPI), with easy installation using pip[2]. These algorithms can be accessed using a simple API which allows users to benefit from their advantages over related algorithms.

3.5 Summary

We have introduced the algorithms CoDA-Regress and CoDA-Cl for supervised learning problems with compositional features. Given a feature vector $\mathbf{X}$ and target vector $\mathbf{y}$, these models fit predictions using a low dimensional representation of the original data. The final loss is computed as the sum of the supervised loss l_y and the reconstruction loss l_x, with the reconstruction loss being scaled by a real valued parameter λ.

[2] `pip install coda-to-end`

Chapter 4

Experimental Methodology

This chapter outlines the methodology we use for our experiments. We provide a detailed explanation of the parameters, data sources, experimental protocols and model tuning which where done to produce the results in 5 and 6. To allow for full transparency and reproducibility, the source code for all experiments is publicly available. The chapter is divided into the following sections:

- Section 4.1. Perhaps the most important part of the methodology is how we evaluate the model. Here we discuss the performance metrics, baseline comparisons and the procedures used to obtain our results.
- Section 4.2 presents the data used for the experiments. We outline the original source and how it is preprocessed, and discuss the resulting features and targets which are used.
- Section 4.3. The final section of this chapter will discuss the methods used to fine tune the model. Specifically, we discuss how the problems of overfitting, numerical stability and convergence were addressed.

4.1 Model Evaluation

K-Fold cross validation is the standard approach for model evaluation in most machine learning contexts. The procedure is as follows: Choose k, the number of groups to split the data into. Shuffle the data, split into k groups and take one as the test set. Train the model on the $k-1$ groups, and evaluate the performance on the test set. Repeat this procedure, changing which group is used for the test set.

This approach is highly unbiased, since the data is shuffled and so trained and tested on separate partitions of the data. In order for the model to perform well overall, it must do well on each partition of the data. This prevents a good model score being due to a lucky initial testing split. Likewise, if a model performs well on only some splits, then this suggests it does not generalise well.

To evaluate our model, we used 4-fold cross validation. To give more certainty in our results, we repeated this 10 times with reshuffled data. The result is a distribution

of the mean cross validation scores over the 10 reshuffled trials. A distribution over the results allows for a better comparison between the models, as it prevents any outliers (good or bad) from confounding the conclusions. For each of the shuffles, we also take a small portion of the data as a validation set. Separate to the testing set, this data is used to diagnose how the model is generalising during training.

4.1.1 Hyperparameter Selection

Here we present the method used to select the hyperparameters when evaluating the model. Figure 4.1 details the grid search methodology used, and how the output of this was used for model evaluation.

For reference, the list of hyperparameters are

- λ: Controls how much to weight the reconstruction loss, as discussed in 3.
- η: The learning rate parameter, controlling the step size in the optimisation algorithm. (2.4)
- n: The number of iterations run for the optimisation.
- d: The feature dimension.
- l: The representation dimension $l < d$, which is used to predict the targets.

The set of parameter values $\{P_i\}_{i=1}^{k}$ was determined by taking all permutations of a fixed list of each parameter. The 3 reshuffled trials were done to minimise the likelihood of a good result being due to chance, in case of a biased shuffle of the data. Once the best set of parameters was found, they were then used to produce the results in 5, using the methodology described above.

4.1.2 Input Dimension

For large feature spaces (which is often the case in microbiome applications) a comparatively small sample size leads to the set of weights being significantly larger than the number of samples. In these cases, there is usually not enough information for network based models to learn any meaningful relationships. To address this, we take a subset of the features to use. We recall from 2 that a subset of a composition is also a composition, so we are justified in applying our algorithm to these. For our experiments, we sort the features by the total number of counts and then truncate at the specified number of features. The amount of features we keep are defined by the hyperparameter d. The dimension of the low level representation is denoted by l.

4.1.3 Metrics

Here we define the specific performance metrics which we use in the cross validation procedure. These metrics are functions of the targets $\mathbf{y}$ and predictions of the model

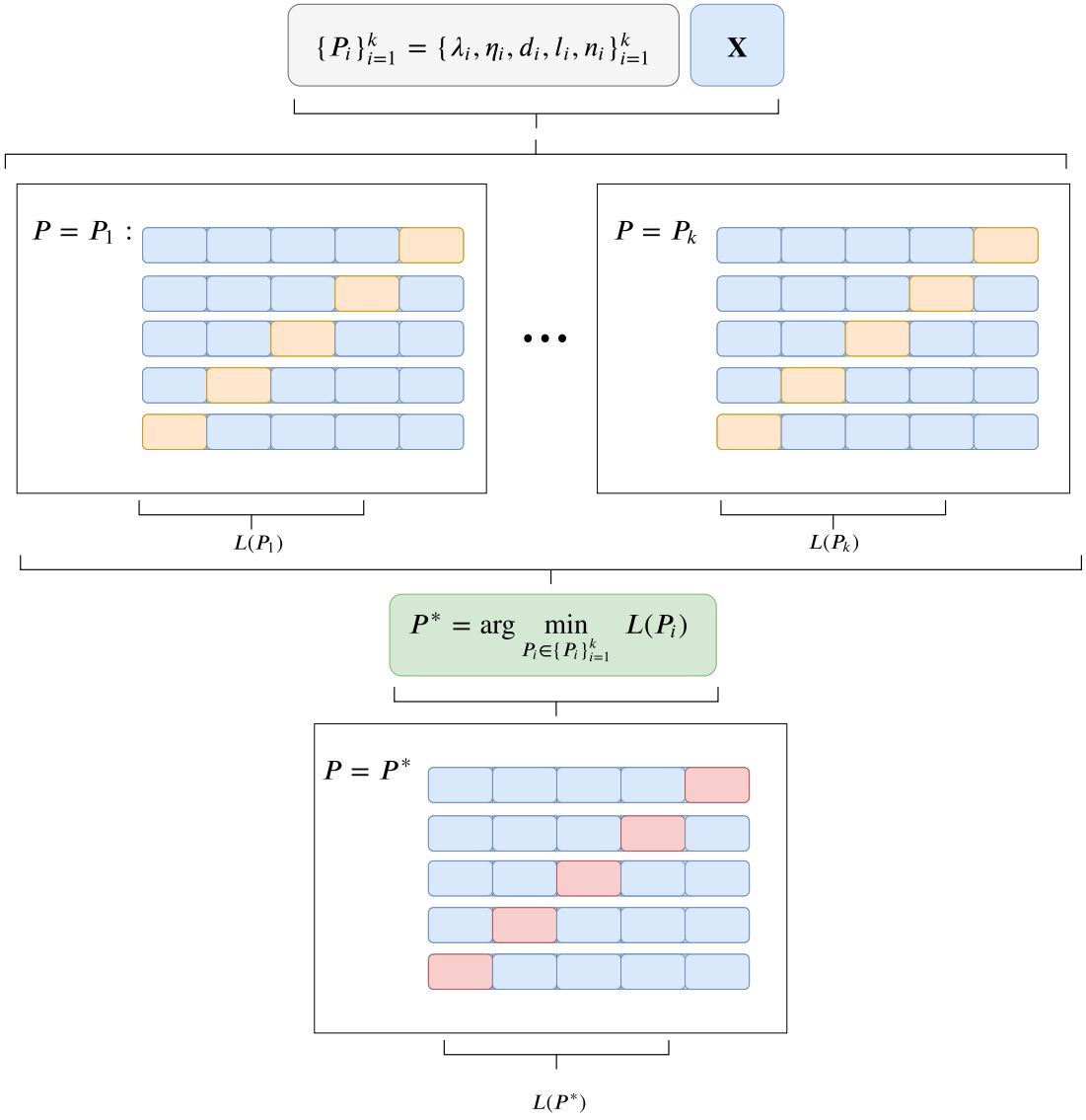


Figure 4.1: Model Evaluation and Hyperparameter Selection: We are given a set of hyperparameters $\{P_i\}_{i=1}^k$ and data $\mathbf{X}$. We fix a partition for $\mathbf{X}$ and obtain the mean cross validation loss $L(P_i)$ for each set of parameters P_i. Doing this 3 times with reshuffled data, we find the optimal hyperparameter set P^*. This is then used to evaluate the model on 10 separate (reshuffled) partitions of $\mathbf{X}$. The final score $L(P^*)$ is given by the mean cross validation score over these trials.

$\hat{\mathbf{y}}$, and are used to assess how close the models predictions are to the targets. Although similar to the loss functions, these metrics are more interpretable and useful in determining the overall performance of the model on a given dataset.

4.1.3.1 Regression: R^2

For Regression problems we use the coefficient of determination, otherwise referred to as the R^2 score [Heinisch, 1962]. The R^2 score is hence defined as a function of real valued targets and predictions, where $\mathbf{y}, \hat{\mathbf{y}} \in R^n$:

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{\sum_{i=1}^{n}(\mathbf{y}_i - \bar{\mathbf{y}})^2}.$$

The standard Mean Squared Error (MSE) score, $\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\mathbf{y}})^2$, is the numerator for the ratio. The denominator represents the total variance in the data, and can also be interpreted as the MSE of a constant valued regressor which predicts the mean. There are several nice properties of this metric. It acts as a measure of the amount of variance explained by the model, with a score of 1 indicating a perfect fit. A score of 0 indicates that the model is doing as well as a model which would only predict the mean. Negative scores indicate even worse performance.

4.1.3.2 Classification: Accuracy

For classification problems, we use the simple metric of classification accuracy. For k-class targets and predictions, $\mathbf{y}, \hat{\mathbf{y}} \in \{0, .., k\}^n$ we define the accuracy as the percentage of correct guesses:

$$Accuracy(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{n}(\mathbf{1}(\mathbf{y}_i = \hat{\mathbf{y}}_i))}{n}. \qquad \text{(where } \mathbf{1}(a = b) = 1 \text{ if } a = b, \text{ 0 otherwise)}$$

The accuracy score is the most commonly used classification metric due to its simplicity and ease of interpretability.

4.1.4 Baselines

To test the performance of our model, we need suitable baseline algorithms to compare against. For regression problems, we refer to Ordinary Least Squares Regression (2.3.1). For classification, we consider softmax based Logistic Regression presented in 2.3.2.2. From these algorithms, we construct the following baseline approaches:

- Naive Regression: This baseline is simply to apply OLS regression directly on the features and targets, in the case of a regression problem
- Naive Logistic Regression: Similarly, this baseline applies logistic regression directly for classification tasks

- CLR + (Logistic) Regression: This baseline applies the CLR transformation, then standard PCA to preprocess the features. The resulting representation is then used as the features for either Regression or Logistic Regression
- CoDA-PCA + (Logistic) Regression: This baseline applies the CoDA-PCA transformation [Avalos et al., 2018], then standard PCA to preprocess the features. The resulting representation is then used as the features for either Regression or Logistic Regression
- Choose Most Frequent: In many classification datasets, there is a non uniform distribution of classes. Thus a model may perform better than random simply by choosing the most frequent class. This baseline corresponds to the score which would be obtained by a model which simply predicts every target to be that which is seen the most in training. This baseline is more indicative of model performance, since if a model can beat this it is more likely to be learning useful properties about the data.

There are a vast number of methods in the literature which are suitable for regression and classification problems. Undoubtedly, many of these would perform better than our baselines. We are more interested in whether our architecture can improve performance by making the representation a function of the targets. The baselines were thus chosen to reflect the loss functions which our model optimises. A difference in performance in comparison to these baselines is therefore more likely to be due to the novel architecture of our model.

4.2 Data Sources

4.2.1 RUG Metagenome Data

The next source was from a recent paper in Nature [Stewart et al., 2019], which provided the metagenome analyses from 283 cattle. The rumen, which is the first stomach in cattle, was the site for the data collection. The resulting metagenome assembled genomes for the rumen (RUGs) form the basis of this dataset. The depth of each genome (which acts as a proxy for species count) is given for each of the cattle in the sample. The collection of depths per cattle is compositional due to the capacity of the sequencing device being limited, as we saw from Gloor et al. [2017]. There were 4941 genomes identified between the 283 cattle. This data provided a good opportunity for testing the model on high dimensional data, since $d >> n$ for dimension $d = 4941$ and sample size $n = 283$.

To preprocess this data, we took the features as the subset of data representing the depth of each genome. We then normalised by the sum over each sample so that the depths represented relative proportions, with all data being on the unit simplex. The targets are the breed of cattle, and were taken from the metadata associated with each

sample. These breeds are Aberdeen Angus, Charolais, Limousin and Luing. Hence we can use this data for testing the CoDA-Cl algorithm.

4.2.2 Atlas Microbiome Data

This dataset [Lahti et al., 2014] contains the profiling of 130 OTUs, taken from the sequencing of 1006 subjects. The subjects were of a Western background, and the sequencing was done on the Human Intestinal Tract (HIT). The OTUs were classified at the genus-like level (i.e. the cut off for sequence similarity will be at the point which roughly distinguishes a genus). This taxonomic profiling was done using a HIT phylogenetic microarray (HITChip), which is based on SSU (16S) rRNA sequencing (see Section 2.5 for an explanation of this).

The data was cleaned to remove any invalid values, resulting in a final sample size of 947. We normalise by the count for each OTU before applying the CoDA methods, so it is embedded on the simplex. There were several potential targets from the metadata, but we use the age of the subjects. This allows us to test the CoDA-Regress algorithm, since the other datasets form classification tasks. For this case, we note the relatively large sample size (n) to feature (d) ratio, with $(d, n) = (130, 947)$. This provides a good comparison to the other datasets, which have far fewer samples.

4.2.3 Dietswap Data

The next dataset [O'Keefe, 2015] comes from a study which compared 130 genus-like OTU abundances between African American and Rural South African subjects. The sequencing was again done on the Human Intestinal Tract and profiled using the HITChip phylogenetic microarray. As a result, the features are identical to the Atlas dataset.

Again, we cleaned the data to remove all invalid values and this gave a usable sample size of 222. Since we have the same $d = 130$ features as in the Atlas example, this will provide an interesting comparison between large and small samples when the features are held constant.

4.3 Model Tuning

4.3.1 Early Stopping

A useful technique in the training of machine learning algorithms is early stopping. This is often employed to reduce model overfitting, as the model weights can become overly optimised on the training data if trained for too long, and so lose their ability to generalise. Early stopping is a simple solution to this; it terminates the training early once the training and validation loss diverge. The model weights are thus chosen at the point when the model has achieved the best generalisation potential. For our implementation, we train for the full number of epochs or until the model has converged

[1]. The weights which give the best results on the validation set are saved, being updated each epoch where there is an improvement. This provides an advantage over traditional early stopping, as an initial drop in validation scores may eventually improve as training progresses.

4.3.2 Numerical Stability

The implementation of Machine Learning algorithms on digital computers often leads to numerical instability. Given that real numbers can only be approximated to a finite precision with floating point numbers, operations on very large or small numbers can lead to numbers beyond the scope of this representation. In addition, the finite precision means that there is an inherent error associated with operations on these numbers. These errors accumulate, and can lead to results which do not align with what they would be analytically. Even when a high precision is used, it is well known that matrix operations are particularly prone to numerical instability. Such operations form the basis for many machine learning algorithms, making numerical stability an important consideration.

Initial testing of our model revealed several of these issues, which manifested as NaN (Not a Number) errors. To fix these problems, we used clamping to restrict the range of input values for several functions. We clamped the denominators in division to be at least as large as 10^{-6}, as denominators close to 0 lead to numbers larger than the precision of floats. Similarly, we clamped exponents to between -30 and 30 to prevent excessively large numbers. Since $\log x \to -\infty$ as $x \to 0$, we again set the minimum value for the inputs to this function as 10^{-6}.

[1]Convergence is achieved when the training losses delta between two epochs is smaller than 10^{-18}.

Chapter 5

Results

In this chapter we outline the results of the experiments using the methodology described in 4. The goal of this chapter is to present evidence to answer the questions we set out to address in the introduction of the thesis:

- Can end-to-end learning give better representations that baseline methods, as judged by the supervised performance?

- Do CoDA based methods improve predictive performance in comparison to traditional (PCA) representations and direct supervised learning?

We structure this chapter to reflect this. We first present the performance comparing each algorithm over all datasets, with varying dimensions. The observations from these plots are useful for answering all of these key questions, and so to avoid any overlap we present them first. Referring to these, we then discuss each question in detail with additional results to support our conclusions for each case. By answering these questions we will show that better representations lead to better predictive performance, and that this appears to be more important as the input dimension increases. We divide the chapter into the following sections:

- Section 5.1 presents the initial results when comparing our model to baseline methods, using the microbiome data in Chapter 4.

- Section 5.2 interprets the results from 5.1 with respect to the end-to-end models CoDA-Cl and CoDA-Regress. We test the hypothesis that our end-to-end architecture can improve performance, and provide further results to elaborate.

- Section 5.3. This section looks at the results in 5.1 to determine whether CoDA based methods can improve performance compared to standard approaches.

5.1 Microbiome Data

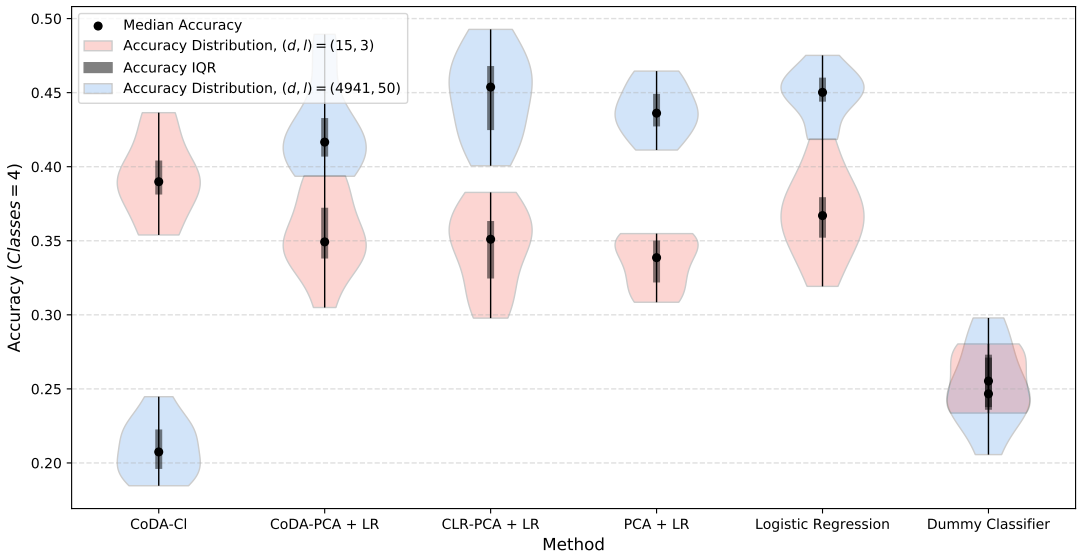


Figure 5.1: CoDA-Cl and Baseline Accuracy on RUG Metagenome data. Accuracy is over 10 trials, for $(d, l) \in \{(15, 3), (4941, 50)\}$. Note the improvement over baselines when $d = 15$, and the near random performance when $d = 4941$

In this section we present the key results from the microbiome experiments, following the methodology introduced in Chapter 4. To briefly reiterate, for each dataset we present a comparison of the predictive performance of the algorithms. The performance is measured by either the classification accuracy or R^2 score, depending on the type of target. The best performing hyperparameters from the grid search described in 4 were used to run 10 reshuffled trials of each algorithm. This was first done using only the dominant 15 taxa, and again using the whole feature space. The distributions in the figures show the mean 4 fold cross validation scores when done over the 10 trials.

The results are shown for the RUG, Diet swap and Atlas data in Figures 5.1, 5.2 and 5.3 respectively. From grid search, we found that the use of larger epochs resulted in better and more consistent results, with overfitting being controlled by early stopping. Hence we used 2000 epochs for these results. The optimal learning rate was found to be constant across all datasets at 10^{-3}, which is what we would expect from the use of the Adam optimisation algorithm. When $d = 15$, we used $\lambda = 10^{-12}$ across all datasets. For the Diet Swap data, we found that $\lambda = 1$ gave the best results on the full data, with $\lambda = 10^{-12}$ being used for the full RUG and Atlas data.

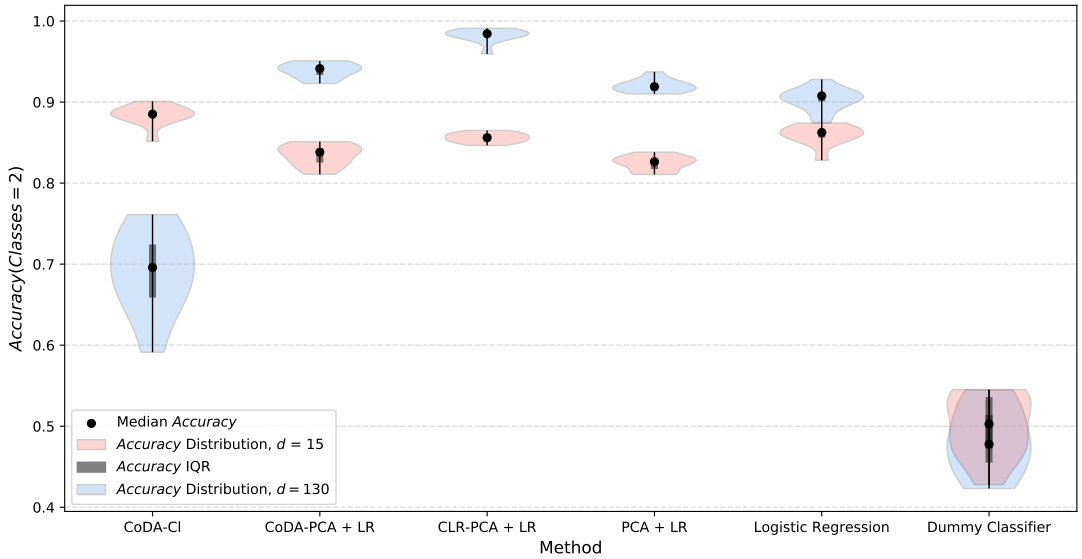


Figure 5.2: CoDA-Cl Accuracy vs Baselines on Diet Swap Microbiome Data

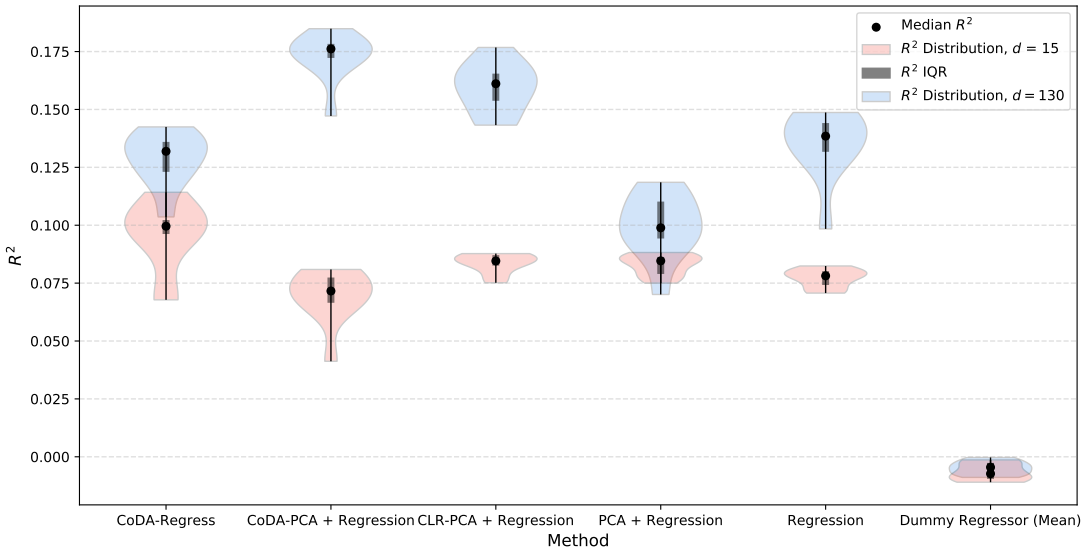


Figure 5.3: CoDA-Cl Accuracy vs Baselines on Atlas Microbiome Data. Note the improvement in performance by CoDA-Cl when $d = 15$ when compared to other representation based methods (PCA,CLR, CoDA-PCA)

5.2 End-to-end: Does it help?

In this section we discuss the performance CoDA-Cl and CoDA-Regress when applied to the microbiome data. We first summarise the important results for these end-to-end models, and follow this with a discussion. We can observe a clear difference in the variance of each algorithm from the plots in 5.1, hence to compare the scores we used Welch's t-test [WELCH, 1947] (as it is implemented in the scipy Python package).

5.2.1 Results Summary

Figure 5.1 showed a significant improvement ($p < 0.05$) for CoDA-Cl in comparison to all representation baselines (CoDA-PCA, CLR and PCA) on the RUG data ($n = 281$) when we use the dominant 15 taxa. Although it has a higher mean score than direct Logistic Regression, the difference was not significant ($p = 0.06$). Hence CoDA-Cl matched the performance of Logistic Regression using the full 15 features, with a representation size of 3. When the full feature space of $d = 4941$ is used, CoDA-Cl drops in performance, and scores below all other baselines ($p < 0.05$).

On the Diet Swap data ($n = 222$), CoDA-Cl significantly outperforms every other method ($p < 0.05$) when $d = 15$. Similarly to the RUG data, the performance drops when $d = 130$. In this case it was not as much of a drop, with it outperforming the dummy baseline.

When applied to the Atlas data ($n = 947$) when $d = 15$, CoDA-Regress showed a significant ($p < 0.05$) improvement over every baseline method. For this case, when we used $d = 130$, CoDA-Regress showed a significant ($p < 0.05$) improvement compared to the $d = 15$ case. It also outperformed PCA + Regression for $d = 130$, but was below the CoDA-PCA and CLR scores ($p < 0.05$)

5.2.2 Discussion

From the results above, it is clear that the end-to-end models consistently perform at or above baseline methods when using the leading 15 taxa. For the RUG data, it outperforms all other representation based methods and matches direct Logistic Regression. For the Atlas and Diet swap data, it surpasses all other methods when $d = 15$.

These results align with our hypothesis that end-to-end models can improve performance. The explanation we propose is that they give representations which allow for a better separation of the targets by the prediction layer. Recall from Chapter 3.1 that a single predictive layer is used to map the latent representation of our end-to-end models to the predictions. Only the prediction loss is being optimised in this layer, as the associated weights do not contribute to the representation. As this loss corresponds to the same loss as (logistic) regression, these models are effectively applying these supervised methods directly to the latent representation. This implies that the only

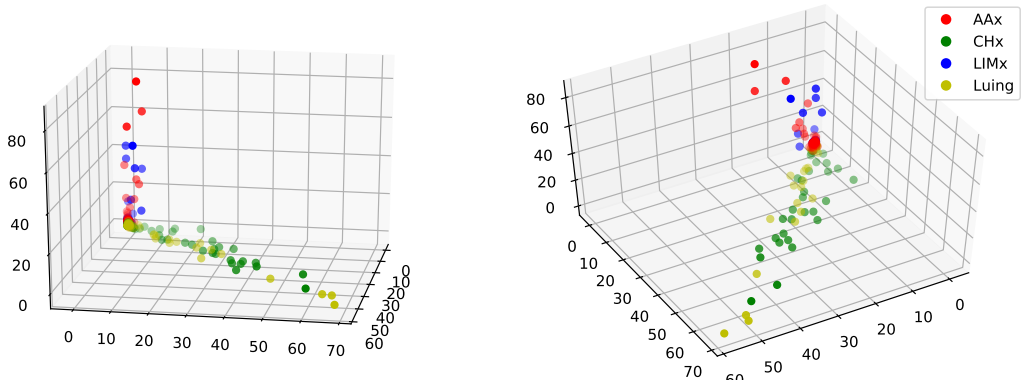
difference between the baselines is the representation. An improvement in performance must indicate that the end-to-end models are giving a representation which is better suited to classifying the targets, which is precisely what motivated the design of the algorithm.

We can show this by plotting the 3 dimensional representation CoDA-Cl gives for the RUG data. Colouring the points based on their class membership allows us to see how easily the classes can be separated. As this forms the input for the classification step, a better separation is more likely to lead to an improvement in accuracy. Figure 5.4 shows the 3D representations for CoDA-Cl using $d = 15, 4941$ and for CoDA-PCA with $d = 15$. When $d = 15$ (Figure 5.4 (a)), the CoDA-Cl representation clearly separates the Charolais (CHx) and Luing cattle from the Angus (AAx) and Limousin (LIMx) cattle. We contrast this with the CoDA-PCA representation in Figure 5.4 (a) (CLR is similar) where there is a larger variance revealed overall, but the classes aren't as visually separable. CoDA-PCA (and CLR/PCA) optimise their representations to preserve the most information from the original data, as it is judged by the reconstruction error. CoDA-Cl is instead optimising a representation which distinguishes the data based on the targets [1]. We note that there is not a complete separation between all four classes, but remember we are only using 3 dimensions out of 4941. Furthermore, the largest mean accuracy was only about 0.45 using the whole 4941 dimensions, which CoDA-Cl comes close to using only 15. From this example, we see that our end-to-end architecture can give representations which better separate the targets. The results show that this is associated with improved predictive performance over baseline methods.

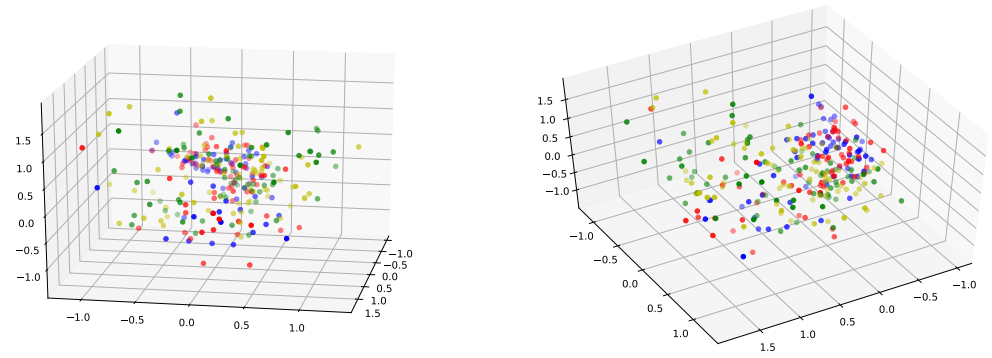
When we use the full feature space, our model does not perform as well. For the RUG case (which we note has the lowest sample size to feature ratio with 281 samples and 4941 features) the model performs about as well as randomly. For the diet swap data, CoDA-Cl performs above the random model but below other baselines. We again note the sample size and feature size, which are 222 and 130 respectively. In the Atlas data, it performs to a similar level as direct Regression, outperforms PCA, and is below CoDA-PCA and CLR. We comment that the Atlas dataset has the largest sample to feature ratio, with 997 samples and 130 features. Clearly, the performance is lower for the cases with fewer samples per feature [2]. The explanation that we suggest is that end-to-end models need more samples to train for a large feature space than baseline methods. Recall from 3.1 that we use a several layer fully connected network for the autoencoder in our models. The number of weights therefore grows at least quadratically in the number of dimensions d (since the first and final layers have d neurons). In contrast, the baseline methods do not optimise any weights, instead giving fixed representations for a given input. For small sample sizes, it is unsurprising that

[1] It is still ensuring the representation is faithful to the original data through the scaling with λ. We compare how this representation can change as λ changes in 6

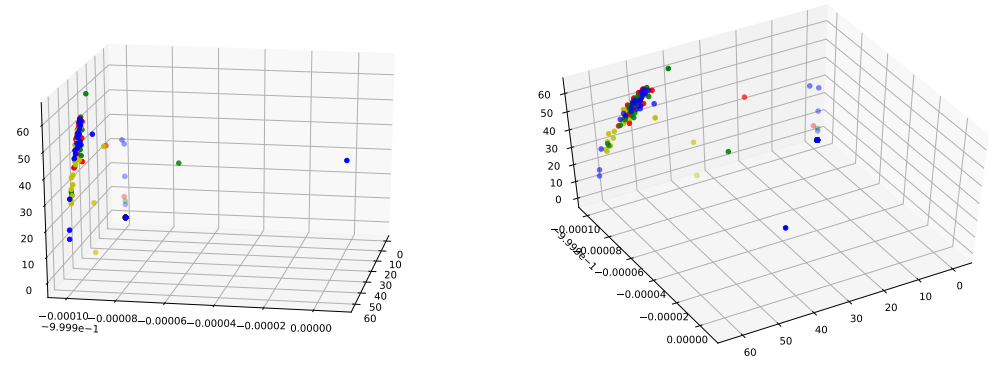
[2] We compare the performance over a larger set of feature sizes using the Diet Swap data in Chapter 6.2, with the same conclusion



(a) CoDA-Cl, for $(d, l) = (15, 3)$

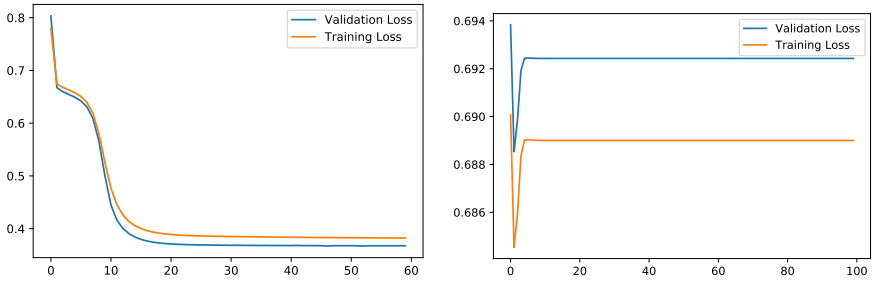


(b) CoDA-PCA, for $(d, l) = (15, 3)$



(c) CoDA-Cl, for $(d, l) = (4941, 3)$

Figure 5.4: 3D representation for CoDA-Cl and CoDA-PCA. Note the difference between the algorithms when $d = 15$: CoDA-Cl better separates the classes, while CoDA-PCA reveals more total variance. When $d = 4941$, no useful representation is found.



(a) (b)

Figure 5.5: Training and Validation loss for CoDA-Cl on Diet Swap Data, $d = 15, 130$

they perform more consistently on high dimensional problems. Figure 5.5 compares the learning curve plots for $d = 15$ and $d = 130$ on the Diet Swap data. At $d = 15$, both the training and validation losses drop rapidly and then plateau, suggesting the model has converged. This is confirmed by the high accuracy scores for this instance. At $d = 130$ these plots clearly show that there is nothing being learned, as we can see from the loss being near constant for the whole training cycle. This lack of learning results in a poor representation, as seen in Figure 5.4 (b). Such a representation does not distinguish between points, hence leading to poor predictive performance. In comparison, looking at the performance on the Atlas regression data our model actually improves as d increases. The feature space is identical to the Diet Swap data, with the same 130 genus-level taxa. There are however more samples per feature, suggesting that the limiting factor in performance is the sample size. The Atlas example suggests that when given enough samples, even for large d the model can perform well.

Our intuition would agree with the claim that an end-to-end model should perform better on high dimensional data. Large feature spaces increase the likelihood of principal components being orthogonal to directions which may explain the target. Under this view, a supervised signal in the loss should enable an end-to-end approach to better find these directions in its representation. It is also possible that this type of model is simply not suited for high dimensional data. Our results suggest sample size is the limiting factor, but these datasets do not cover a wide enough range of examples to reasonably confirm this. It would be informative to test this hypothesis on a large sample (e.g. $n > 10^4$) microbiome dataset. If sample size is indeed the limiting factor, then we would expect to see an improvement over all baselines as we did for the $d = 15$ case in a high dimensional setting.

In summary, we see that the end-to-end approach can outperform baseline methods when taking the dominant 15 taxa. We saw that this was due to a representation which better separates the targets. This supports the concept of using the targets to construct the representation. As the dimensions increase, we found that the performance of our model drops. We argue that poor performance in the high dimensional case is most likely explained by a lack of samples. As we saw from the Atlas dataset which had the

largest sample size to feature ratio, the model can still perform at the level of baseline methods in the high dimensional setting.

5.3 CoDA Representations: Are they better?

In this section we compare the performance of CoDA based representations (CoDA-PCA, CLR-PCA) to standard PCA and direct (Logistic) Regression. We omit the end-to-end models here as we discussed those in detail in the previous section. As before, we summarise the relevant results for this research question, and then proceed to discuss them with additional support for our hypotheses.

5.3.1 Results Summary

From the RUG dataset, the performance of all representation based methods was similar. The performance increased across all algorithms similarly as d increased, with all methods performing closely. This dataset formed the most challenging problem due to the small sample size and large feature space. The small increase in performance from $d = 15$ to $d = 4941$ suggests that either a small subset of the features contained useful information, or that there were not enough samples. From this dataset, there is insufficient evidence to distinguish the algorithms.

For the Diet swap data, CLR outperformed CoDA-PCA with a significant difference in group means ($p < 0.05$) between the two. Both methods outperformed PCA and Logistic Regression ($p < 0.05$). There was a less pronounced difference when $d = 15$, with only the difference between CLR and PCA being significant ($p < 0.05$). Hence for this dataset, we conclude that the CoDA based methods were superior when $d = 130$.

From the Atlas data, there is not much difference between the algorithms when $d = 15$. When $d = 130$, we note a significant difference between CoDA-PCA and CLR ($p < 0.05$), with CoDA-PCA performing better. Both PCA and direct Regression are significantly ($p < 0.05$) outperformed by both CoDA-PCA and CLR-PCA. Again, we can therefore conclude that the CoDA based methods were superior for $d = 130$.

5.3.2 Discussion

As we stated, there was not enough evidence to distinguish the algorithms for the RUG data. For the other 2 datasets, we saw that the CoDA based methods clearly performed better when the whole feature set of 130 was taken. This suggests that using these methods is more important for larger dimensions. Although these are not using the targets for the representation (like in the end-to-end case), these methods still outperform PCA and direct (Logistic) Regression. Since all methods are followed with Logistic Regression, it follows that the CoDA methods are giving better representations for supervised learning. We can illustrate this in a similar way as the end-to-end case. Figure 5.7 shows the 1D, 2D and 3D representations for the Atlas data given CoDA-PCA, CLR-PCA and PCA. We first note the similarity between CoDA-PCA and CLR.

The important point to observe here is the amount of variance we observe in each case. For the 1D case, PCA clusters most of the data in half of the space. While CoDA-PCA and CLR split the data more uniformly, it is still hard to distinguish any points based on their target. In the 2D case, CoDA and CLR spread the data fairly evenly over the plane. We can now see a rough split between the points based on the target (Age), although there is a large overlap. This is not the case with the PCA representation, which again clusters the points in a small section of the space. The final 3D representation for CoDA and CLR further splits the data, making it easier to distinguish points based on the target. The PCA representation is far worse, with comparatively very little spread between the data (although we can now see a rough split between old and young subjects). It may be argued that there is still a large degree of overlap even in the CoDA cases, but this is a difficult problem. We do not expect to be able to fully explain the age of the subjects based on a small genus level microbiome sample. Any noticeable trend is good, and we can clearly see that the CoDA methods reveal more variance within the data. This variance can make it easier for a supervised learning algorithm to distinguish between points, which leads to improved performance. This example suggests that the compositional structure of the data may be more important for increasing dimensions.

To investigate this claim more empirically, we compare the R^2 scores between these algorithms when applied to the Atlas dataset. We can see from Figure 5.6 that the performance of PCA is lower than both CLR and CoDA-PCA for the majority of cases. In addition to this, the performance of PCA dropped as the dimensions grew, with the performance of CoDA-PCA and CLR-PCA increasing. Furthermore, both CoDA-PCA and CLR-PCA outperform direct Regression.

To summarise, we have seen in this section that CoDA-PCA and CLR-PCA give us better representations in terms of supervised performance that PCA and direct supervised learning. This complements the results from [Avalos et al., 2018], which showed they preserved structure better than standard PCA. In the context of our first research question, these results suggest that CoDA based methods improve predictive performance in comparison to standard PCA and direct supervised learning.

5.4 Summary

In this chapter we have presented the results from applying our end-to-end model and baseline methods on 3 Microbiome datasets. For all 3 datasets, our end-to-end model performed significantly above baseline when using the 15 most frequent taxa. By visualising the representation, we showed that this can be attributed to a representation which better separates the targets. The performance decreased as the dimensions increased, which we argued is a result of insufficient data for the models to learn. Addressing our research question, these results provide evidence to suggest that our end-to-end model can improve performance over baselines, under the assumption that there is a sufficient sample size to input dimension ratio. With that being said, there is still the

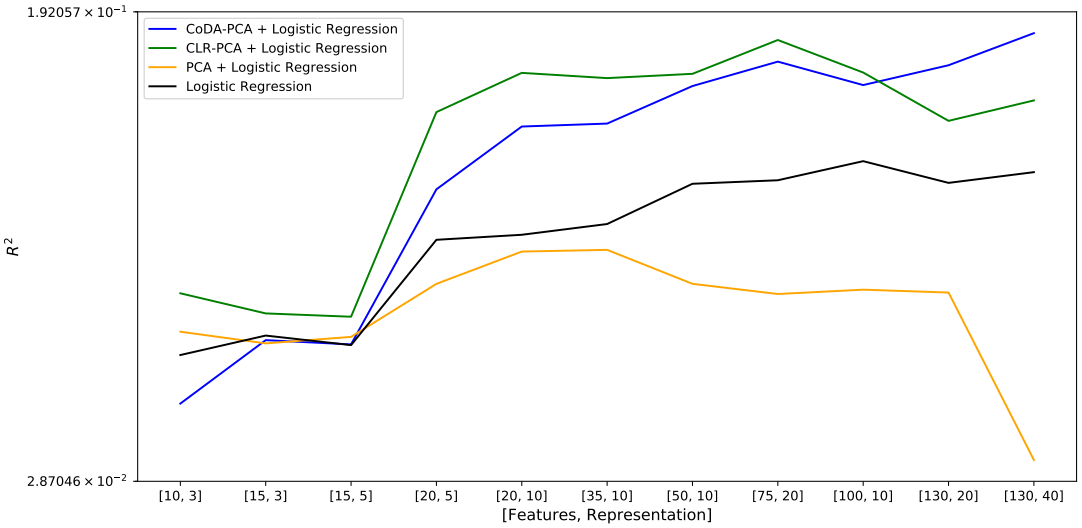
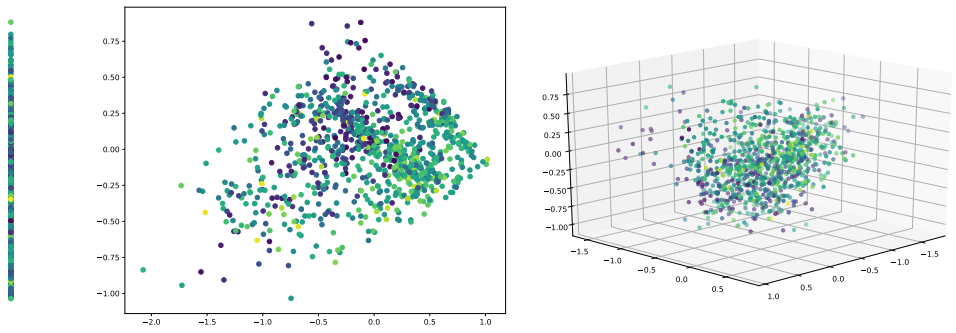
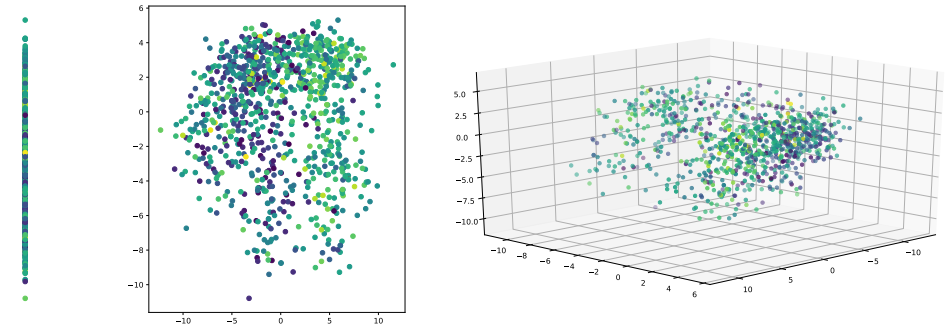


Figure 5.6: R^2 for CoDA-PCA, CLR-PCA, PCA and Logistic Regression on Atlas data, with varying Feature size d and latent Dimension l. CoDA-PCA and CLR-PCA improve significantly as the dimensions increase, and consistently oupertform direct Regression. PCA improves far slower, then plateaus with a sharp drop in performance for $d = 130$.

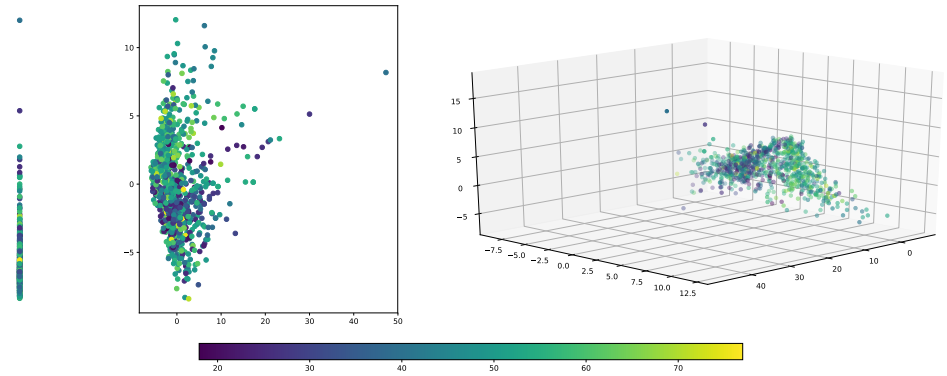
possibility that the model is not suited for high dimensional data, and more testing is needed to confirm this. For the other research question, we showed the superiority of CoDA based methods over both PCA and direct supervised learning, particularly in the high dimensional setting. The application of these traditional methods is still commonplace, even in the high dimensional application of microbiome analysis. These results add to the current literature on the importance of CoDA based methods. We show that for high dimensional applications such as microbiome studies, CoDA based representations can improve performance for supervised learning methods.



(a) CoDA-PCA



(b) CLR-PCA



(c) PCA

Figure 5.7: 1D, 2D and 3D Representations for Atlas Microbiome Data, coloured based on Subject Age

Chapter 6

Hyperparameter Sensitivity

In this chapter we investigate the effect of each of the hyperparameters on our model. The results from this Chapter were used to select the choice of hyperparameters for the results in 5. We separate this chapter to explore each hyperparameter in depth, which is done as follows:

- Section 6.1 looks at the effect that the scaling parameter λ can have on the underlying representation produced by our model.
- Section 6.2 investigates the effect of the input size and latent dimensions on predictive performance.
- Section 6.3 looks at the relationship between λ and the input dimension, and how this can influence the performance of the model.

6.1 The effect of λ on Representation

Recall the definition of the end to end loss in 3.4: Given a reconstruction loss l_x (e.g. CoDA-PCA) and supervised loss l_y (e.g. Least Squares), the end to end loss $l_{end-to-end}$ is defined as

$$l_{end-to-end} = l_y + \lambda l_x.$$

As we mentioned in 3, λ acts as a control mechanism to weight the importance of the reconstruction. Hence we expect to see the representation of a high λ model to resemble the representation given by just optimising l_x. As λ is reduced, this would potentially change to favour representations which are more influential on the supervised loss l_y.

To test this, we took an example Aitchinson dataset [1], which has 39 samples and a feature dimension of 3. We then applied CoDA-PCA and CoDA-Regress to obtain 2D representations. We ran CoDA-Regress with a large and small setting for λ. Figure 6.1 shows the corresponding 2D representations produced. The different colours

[1] Data 5. Sand, silt, clay compositions of 39 sediment samples at different water depths in an Arctic lake

correspond to the same group of points between the plots, allowing us to compare the similarity between the clusters. We can see that the small λ setting (left, subfigure (b)) leads to a far less structured representation. It has preserved the rough ordering of the clusters produced by CoDA-PCA (subfigure (a)), but has failed to maintain the variance between the points. The large λ setting (right, subfigure (b)) more closely resembles what is given by CoDA-PCA. This result experimentally supports the hypothesis that larger λ settings gives similar representations as CoDA-PCA. This is not surprising, however it is a good indication that the model is working as we expect it to.

6.2 The effect of Feature and Representation Size on predictive performance

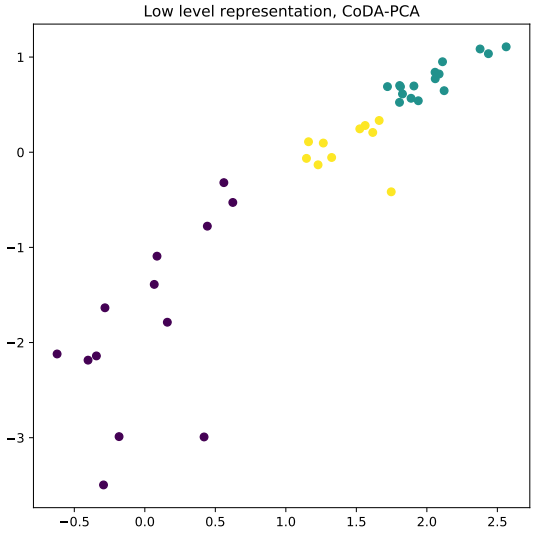
In this section we see look into how the feature size d and latent dimension l influence the performance of the end-to-end model. We will use the Diet swap Microbiome dataset to do this, comparing the R^2 score for CoDA-Cl as it varies based on the dimensions. The results of this are shown in Figure 6.2

For the smaller feature sizes up to 15, the model performs well with approximately 0.85 accuracy. As this increases to 20 features, it drops to between 0.7 and 0.75. For features larger than 20, the model consistently has an accuracy around 0.55 which is random. From Chapter 5, the model does not seem to learn for large feature sizes, if there are not enough samples. Here we can see here that this forms a trend for a wider range of values, with the performance decreasing as the dimensions increase. The rate of decrease is linked to the number of samples, as we saw with the Atlas data (which has significantly more samples) where CoDA-Regress performed better for $d = 130$ despite using the same feature space as this example. As we discussed in Chapter 5, this is most likely a result of the sample size to feature ratio. For this example, we have a fixed sample size and so this ratio decreases (along with performance) as the features. We note that the causality here is speculative, and although there is some evidence to support it (e.g. the training curves), we cannot confirm this with full certainty. The alternative would be that end-to-end models are simply not suited to high dimensional data. This seems unlikely, when looking at the improvement seen by CoDA-Regress on the Atlas data, which had the same features as this example but a larger number of samples.

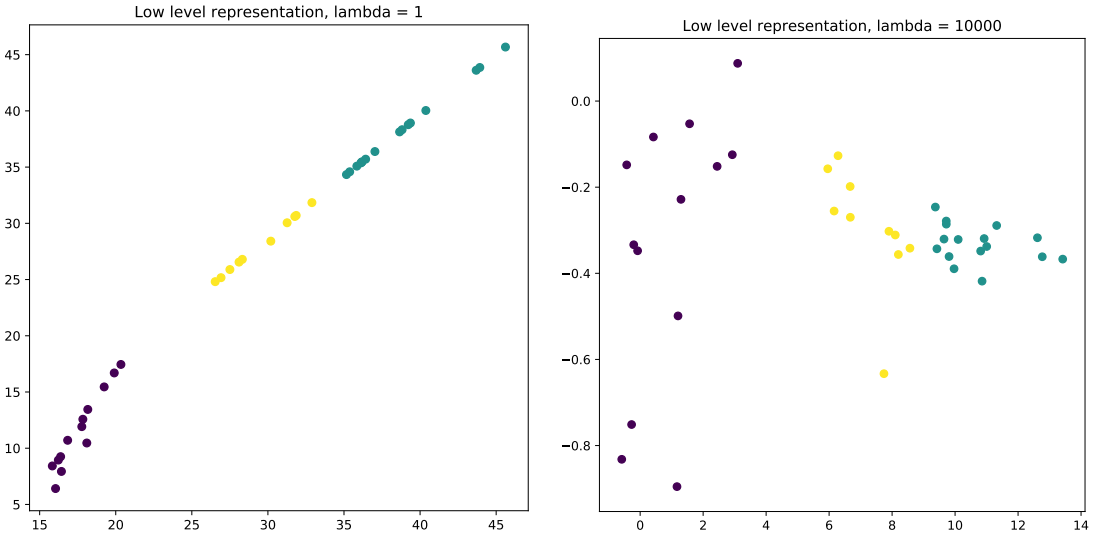
6.3 The interaction of λ and Dimension

In this section we will investigate how the scaling parameter λ interacts with the input and latent dimensions (d, l) as judged by predictive performance.

Table 6.1 shows the average performance of CoDA-Cl when run on the Diet Swap data for varying choices of λ and (d, l), with all else constant. We have the following observations:



(a) CoDA-PCA Representation



(b) CoDA-Regress Representation for $\lambda = 1$ (left), $\lambda = 10^4$ (right)

Figure 6.1: The effect of λ on representation

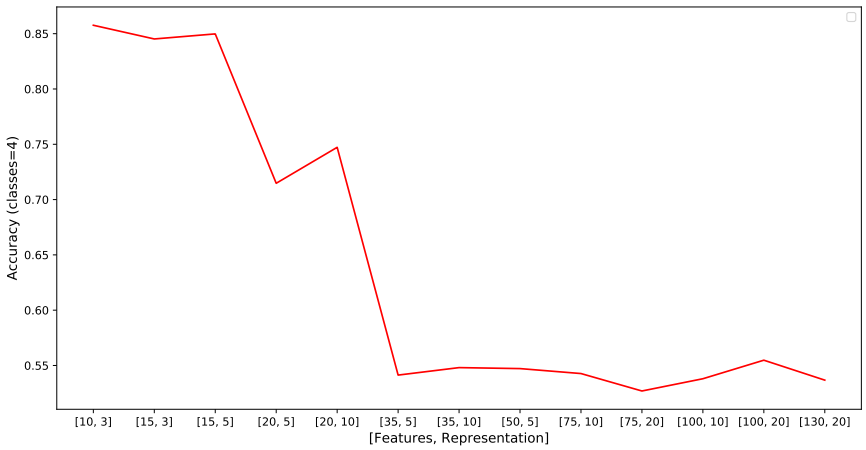


Figure 6.2: Accuracy for CoDA-Cl on Diet Swap Data, for varying Feature size d and latent Dimension l

	λ								
(d, l)	0	10^{-12}	10^{-10}	10^{-8}	10^{-5}	10^{-2}	0.5	1	5
(10, 5)	0.88	**0.89**	0.88	**0.89**	**0.89**	0.84	0.85	0.82	0.84
(15, 5)	0.88	0.88	**0.89**	**0.89**	**0.89**	0.85	0.83	0.85	0.83
(20, 10)	**0.88**	0.86	0.86	0.85	0.83	0.69	0.74	0.68	0.71
(50, 10)	**0.65**	0.61	0.53	**0.65**	0.61	0.59	0.63	0.58	0.64
(100, 20)	0.52	0.61	0.55	0.61	**0.70**	0.69	0.69	0.68	0.67
(130, 20)	0.56	0.55	0.53	0.62	**0.68**	0.63	**0.68**	**0.68**	0.65

Table 6.1: CoDA-Cl Classification Accuracy on Dietswap data for λ and (d, l)

- The classification accuracy is the greatest for smaller values of (d, l) . For all $d < 20$, the best accuracy is above 0.88. For $d \geq 50$ The performance drops significantly to between 0.65 and 0.7. This is a similar trend to what we observed in Chapter 5 and is due to there being insufficient information for the model to learn.

- As the dimensions increase (over 100) the larger λ scores do far better (though still poorly in comparison). Small to mid λ does best for small d, with comparatively little variation. Experimentally, from Chapter **??** we see that CoDA based representations perform better as d increases. A larger λ ensures the representation will align with the CoDA-PCA equivalent, as we saw earlier in the Chapter (6.1).

6.4 Summary

In this Chapter we explored the effect of the hyperparameters on our end-to-end model. A summary of our findings in each section are:

- In section 6.1 we experimentally show that λ is tuning the model as expected, with larger values giving similar representations to CoDA-PCA.

- In 6.2, we elaborate on the observations in Chapter 5 and show that model performance decreases as the dimensions increase.

- Section 6.3 found larger λ values to perform better on large dimensions, which aligned with expectations.

Chapter 7

Conclusion

For this thesis we looked at representation methods for compositional data, comparing how they perform when applied to supervised learning tasks. To test these methods, we used three microbiome datasets where each represented a different challenge. By following the experimental procedure which we outlined in Chapter 4, we used these results to answer the two research questions motivating this thesis.

For the first question, we asked whether CoDA-PCA and CLR-PCA give representations which outperform standard PCA and direct supervised learning. From Chapter 5, our results on the microbiome data lead us to conclude that this is indeed the case. For 2 out of 3 datasets, both CoDA-PCA and CLR-PCA showed statistically significant ($p < 0.05$) improvement when compared to PCA and direct supervised learning. We can consider these results to empirically ground the results from Avalos et al. [2018] which show the superiority of CoDA-PCA to PCA. Further investigation revealed that this improvement in performance appears to increase with the dimensions, suggesting that high dimensional data would suffer more by ignoring the compositional structure. The scope of microbiome data can only grow as the technology becomes cheaper and more readily available. This finding shows that the use of appropriate methods for such large scale data is imperative. To this end, our results accentuate the current consensus on best practice and highlight the importance of theoretically grounded models in this area.

Our second question was regarding the performance of the end-to-end model. We formally defined this model in Chapter 3.1, motivating its design by the independence between representations of other methods and the targets. We therefore expected to see an improvement in performance over baseline methods, with the representation being a function of the targets. Testing on the microbiome datasets, we found the performance of this model to improve with the number of samples per feature. In comparison to baseline methods, we showed the model can produce representations which better partition the data based on the prediction targets. We found there to be superior predictive performance in these cases. For the other cases, we argued that the sample size was the limiting factor in performance. To support this, we tested the model over a wide range of dimensions. The results of this showed an inverse relationship between dimensions and performance, for a fixed number of samples. Given

that these results were only seen over a few datasets, we are however reluctant to draw any final conclusions on this point. We conducted further experiments on the model parameters in Chapter 6, where we found the scaling parameter λ to regularise the loss as we expected, with high λ losses giving representations similar to CoDA-PCA. We subsequently found that higher λ parameters lead to better performance in high dimensions, when we compare to models which are otherwise identical. Returning to the question posited in Chapter 1, for high dimensional data we have seen there is insufficient evidence to give a definitive answer to this question. In low dimensional cases, our end-to-end model clearly gives representations which better partition the data in comparison to baselines, which answers the question positively in this case. The promising performance for the low dimensional case is cause for optimism. As the availability of large sample microbiome data increases, we will be able to properly test whether this improvement continues to higher dimensions.

7.1 Future Work

There is a large avenue for future research in this area. As we concluded with, from the experiments we ran there is no real indication of how the model may perform in the high dimensional, high sample setting. The most obvious avenue for future work is to find or curate an appropriately large dataset where this can properly be tested.

With regards to an end-to-end model, we stated in Chapter 3 that the concept can be applied to any choice of representation and supervised loss. Hence we could use any other representation error, with the immediate choice being the PCA reconstruction loss. The additive combination of the loss, as $l_x + \lambda l_y$, is also somewhat arbitrary. There are potentially other perspectives to approach similar constrained optimisation problems.

There is of course more work to be done for Compositional Data Analysis in general, with the first step being to raise awareness that standard Euclidean methods do not work well on the simplex. That being said, there has been significant progress in this when considering e.g. Gloor et al. [2017].

Bibliography

Aitchison, J., 1982. The statistical analysis of compositional data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 44, 2 (1982), 139–177. http://www.jstor.org/stable/2345821. (cited on pages 2, 8, and 12)

Aitchison, J., 2003. A concise guide to compositional data analysis. *2nd Compositional Data Analysis Workshop; Girona, Italy.*, (2003). http://www.leg.ufpr.br/lib/exe/fetch.php/pessoais:abtmartins:a_concise_guide_to_compositional_data_analysis.pdf. (cited on pages 3 and 8)

Avalos, M.; Nock, R.; Ong, C. S.; Rouar, J.; and Sun, K., 2018. Representation learning of compositional data. In *Advances in Neural Information Processing Systems 31* (Eds. S. Bengio; H. Wallach; H. Larochelle; K. Grauman; N. Cesa-Bianchi; and R. Garnett), 6679–6689. Curran Associates, Inc. http://papers.nips.cc/paper/7902-representation-learning-of-compositional-data.pdf. (cited on pages 2, 3, 8, 9, 13, 15, 32, 34, 39, 51, and 61)

Boissonnat, J.-D.; Nielsen, F.; and Nock, R., 2010. Bregman voronoi diagrams. *Discrete & Computational Geometry*, 44, 2 (Sep 2010), 281–307. doi:10.1007/s00454-010-9256-1. https://doi.org/10.1007/s00454-010-9256-1. (cited on page 13)

Boyd, S. and Vandenberghe, L., 2009. *Principal Component Analysis,Second Edition*. Cambridge University Press. (cited on page 11)

Bragg L, T. G., 2014. Metagenomics using next-generation sequencing. *Methods Mol Biol*, 1096, 1 (2014), 183–201. doi:10.1146/annurev-statistics-010814-020351. (cited on pages 26 and 27)

Collins, M.; Dasgupta, S.; and Schapire, R. E., 2002. A generalization of principal components analysis to the exponential family. In *Advances in Neural Information Processing Systems 14* (Eds. T. G. Dietterich; S. Becker; and Z. Ghahramani), 617–624. MIT Press. http://papers.nips.cc/paper/2078-a-generalization-of-principal-components-analysis-to-the-exponential-family.pdf. (cited on page 12)

Csáji, B. C., 2001. *Approximation with Artificial Neural Networks*. Master's thesis, Faculty of Sciences, Eötvös Loránd University, Hungary. (cited on page 20)

Forsythe, G. E. and Moler, C. B., 1967. *Computer solution of linear algebraic systems*. Prentice Hall. (cited on page 11)

Gloor, G. B.; Macklaim, J. M.; Pawlowsky-Glahn, V.; and Egozcue, J. J., 2017. Microbiome datasets are compositional: And this is not optional. *Frontiers in Microbiology*, 8 (2017), 2224. doi:10.3389/fmicb.2017.02224. https://www.frontiersin.org/article/10.3389/fmicb.2017.02224. (cited on pages 2, 26, 39, and 62)

Heinisch, O., 1962. Steel, r. g. d., and j. h. torrie: Principles and procedures of statistics. (with special reference to the biological sciences.) mcgraw-hill book company, new york, toronto, london 1960, 481 s., 15 abb.; 81 s 6 d. *Biometrische Zeitschrift*, 4, 3 (1962), 207–208. doi:10.1002/bimj.19620040313. https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.19620040313. (cited on page 38)

Howley, T.; Madden, M. G.; O'Connell, M.-L.; and Ryder, A. G., 2006. The effect of principal component analysis on machine learning accuracy with high-dimensional spectral data. *Knowledge-Based Systems*, 19, 5 (2006), 363 – 370. doi:https://doi.org/10.1016/j.knosys.2005.11.014. http://www.sciencedirect.com/science/article/pii/S0950705106000141. AI 2005 SI. (cited on page 21)

Janda, J. M. and Abbott, S. L., 2007. 16s rrna gene sequencing for bacterial identification in the diagnostic laboratory: Pluses, perils, and pitfalls. *Journal of Clinical Microbiology*, 45, 9 (2007), 2761–2764. doi:10.1128/JCM.01228-07. https://jcm.asm.org/content/45/9/2761. (cited on page 26)

Joliffe, I., 2002. *Principal Component Analysis,Second Edition*. Springer. (cited on pages 9, 10, and 11)

Kingma, D. P. and Ba, J., 2014. Adam: A method for stochastic optimization. (cited on pages 23 and 34)

Kramer, M. A., 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37, 2 (1991), 233–243. doi:10.1002/aic.690370209. https://aiche.onlinelibrary.wiley.com/doi/abs/10.1002/aic.690370209. (cited on page 21)

Lahti, L.; Salojärvi, J.; Salonen, A.; Scheffer, M.; and de Vos, W. M., 2014. Tipping elements in the human intestinal ecosystem. *Nature Communications*, 5, 1 (Jul. 2014), 4344. https://doi.org/10.1038/ncomms5344. (cited on page 40)

Leon, S.; Bjorck, A.; and Gander, W., 2013. Gram-schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20 (05 2013). doi:10.1002/nla.1839. (cited on page 11)

Li, H., 2015. Microbiome, metagenomics, and high-dimensional compositional data analysis. *Annual Review of Statistics and Its Application*, 2, 1 (2015), 73–94. doi:10.1146/annurev-statistics-010814-020351. (cited on page 26)

Liu, R.; Kuang, J.; Gong, Q.; and Hou, X., 2003. Principal component regression analysis with spss. *Computer Methods and Programs in Biomedicine*, 71, 2 (2003), 141 – 147. doi:https://doi.org/10.1016/S0169-2607(02)00058-5. http:

//www.sciencedirect.com/science/article/pii/S0169260702000585. (cited on page 17)

Mackiewicz, A. and Ratajczak, W., 1993. Principal components analysis (pca). *Computers & Geosciences*, 19, 3 (1993), 303 – 342. doi:https://doi.org/10.1016/0098-3004(93)90090-R. http://www.sciencedirect.com/science/article/pii/009830049390090R. (cited on page 9)

McDonald, D.; Hyde, E.; Debelius, J. W.; Morton, J. T.; Gonzalez, A.; Ackermann, G.; Aksenov, A. A.; Behsaz, B.; Brennan, C.; Chen, Y.; DeRight Goldasich, L.; Dorrestein, P. C.; Dunn, R. R.; Fahimipour, A. K.; Gaffney, J.; Gilbert, J. A.; Gogul, G.; Green, J. L.; Hugenholtz, P.; Humphrey, G.; Huttenhower, C.; Jackson, M. A.; Janssen, S.; Jeste, D. V.; Jiang, L.; Kelley, S. T.; Knights, D.; Kosciolek, T.; Ladau, J.; Leach, J.; Marotz, C.; Meleshko, D.; Melnik, A. V.; Metcalf, J. L.; Mohimani, H.; Montassier, E.; Navas-Molina, J.; Nguyen, T. T.; Peddada, S.; Pevzner, P.; Pollard, K. S.; Rahnavard, G.; Robbins-Pianka, A.; Sangwan, N.; Shorenstein, J.; Smarr, L.; Song, S. J.; Spector, T.; Swafford, A. D.; Thackray, V. G.; Thompson, L. R.; Tripathi, A.; Vázquez-Baeza, Y.; Vrbanac, A.; Wischmeyer, P.; Wolfe, E.; Zhu, Q.; ; and Knight, R., 2018. American gut: an open platform for citizen science microbiome research. *mSystems*, 3, 3 (2018). doi:10.1128/mSystems.00031-18. https://msystems.asm.org/content/3/3/e00031-18. (cited on pages 2 and 23)

Nguyen, N.-P.; Warnow, T.; Pop, M.; and White, B., 2016. A perspective on 16s rrna operational taxonomic unit clustering using sequence similarity. *npj Biofilms and Microbiomes*, 2, 1 (Apr. 2016), 16004. https://doi.org/10.1038/npjbiofilms.2016.4. (cited on page 26)

O'Keefe, S. J. D. e. a., 2015. Fat, fibre and cancer risk in african americans and rural africans. *Fat, fibre and cancer risk in African Americans and rural Africans*, (2015). https://doi.org/10.1038/ncomms7342. (cited on page 40)

Qu, K.; Guo, F.; Liu, X.; Lin, Y.; and Zou, Q., 2019. Application of machine learning in microbiology. *Frontiers in Microbiology*, 10 (2019), 827. doi:10.3389/fmicb.2019.00827. https://www.frontiersin.org/article/10.3389/fmicb.2019.00827. (cited on page 2)

Shlens, J., 2005. A tutorial on principal component analysis. *Systems Neurobiology Laboratory, Universityof California at San Diego*, 82 (2005), 2224. https://www.cs.cmu.edu/~elaw/papers/pca.pdf. (cited on page 9)

Stewart, R. D.; Auffret, M. D.; Warr, A.; Walker, A. W.; Roehe, R.; and Watson, M., 2019. Compendium of 4,941 rumen metagenome-assembled genomes for rumen microbiome biology and enzyme discovery. *Nature Biotechnology*, 37, 8 (Aug. 2019), 953–961. https://doi.org/10.1038/s41587-019-0202-3. (cited on pages 2 and 39)

WELCH, B. L., 1947. THE GENERALIZATION OF 'STUDENT'S' PROBLEM WHEN SEVERAL DIFFERENT POPULATION VARLANCES ARE INVOLVED. *Biometrika*, 34, 1-2 (01 1947), 28–35. doi:10.1093/biomet/34.1-2.28. https://doi.org/10.1093/biomet/34.1-2.28. (cited on page 46)

Zaresky-Williams, E., 2019. An algorithm for approximating continuous functions on compact subsets with a neural network with one hidden layer. (02 2019). (cited on page 20)