Scalable Manifold Learning and Related Topics

James McQueen

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Marina Meilă, Chair

Hari Narayanan

Yen-Chi Chen

Program Authorized to Offer Degree:
Department of Statistics

University of Washington

Abstract

Scalable Manifold Learning and Related Topics

James McQueen

Chair of the Supervisory Committee:
Professor Marina Meilă
Department of Statistics

The subject of manifold learning is vast and still largely unexplored. As a subset of unsupervised learning it has a fundamental challenge in adequately defining the problem but whose solution is to an increasingly important desire to understand data sets *intrinsically*. It is the overarching goal of this work to present researchers with an understanding of the topic of manifold learning, with a description and proposed method for performing manifold learning, guidance for selecting parameters when applying manifold learning to large scientific data sets and together with open source software powerful enough to meet the demands of big data.

First we describe the topic of manifold learning in the context of manifolds and Riemannian metrics. We use this framework to define a loss function which encodes deviation from an isometry into *Euclidean* space. The loss function we define explicitly handles the case where the embedding dimension is larger than the intrinsic dimension. By doing so it ensures that the resulting embedding will still be a submanifold of the original intrinsic dimension, this is a significant departure from previous methods. Due to the iterative nature of the algorithm `RiemannianRelaxation` it naturally scales to large data sets.

Second we provide a cohesive overview of several heuristics for selecting parameters when performing manifold learning, the bandwidth and dimension parameters. We demonstrate how to scale these methods to deal with the demands of big scientific data sets and apply

these methods to a large astronomical database of galaxy spectra. Third, we combine all of these methods into an open source python package, `megaman`, which is explicitly designed with the challenges of research in mind: dealing with large data sets, selecting parameters, repeating procedures and storing interim steps. With these together it is our hope to provide scientists and researchers alike with the tools to apply manifold learning. As an additional, related topic, we discuss spectral clustering. We overview the subject and how it relates to manifold learning as well as propose a scalable (online) pre-processing algorithm for pruning the graph before performing spectral clustering. We then demonstrate this using a large 1.8M paper citation network from Jstor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

The author would like to express his sincere gratitude to his advisor Professor Marina Meilă for her guidance and advice, Professors Hari Narayanan, Yen-Chi Chen, Carlos Guestrin, and John Lee for their insight and comments, Professor Michael Perlman for his support, and his family for their continued encouragement.

DEDICATION

to my beloved wife, Kira

Chapter 1

INTRODUCTION

With massively accelerating rates of data recording across all fields of research it is increasingly important to understand large data sets *intrinsically* without respect to some set of labels which are often expensive (both in terms of time and money) to create. The area of *unsupervised* learning is therefore an important field of study. The size of data sets is scaling both in the number of individual observations (whether it be galaxies in the sky, phrase in the English language, or images in a data base) but also in the *dimension*, number of features per observation, of each observations (the resolution of the spectra, the length of the phrase, the number of pixels in the images). It is often the case in statistics and machine learning that the number of observed features per data point (often called the dimension of the data) is very large.

In many of these examples while the observed dimension of the data is very large there may only be a small amount variation between dimensions. In some sense the data are truly of lower dimension but are being observed in a higher dimensional space. In these cases it is advantageous to apply a dimension reduction technique to the data to both reduce the size of the data set and potentially increase performance on some other task or model, be it scientific inquiry or supervised learning.

Manifold learning, non-linear dimension reduction algorithms, has been applied in many different areas including face recognition [60], speaker verification [3], chemistry [44], and astronomy [55]. Specifically, throughout this thesis we consider several real world as well as synthetic data sets in order to demonstrate the performance and application of manifold learning models discussed in this thesis.

1.1 Synthetic Data Sets

Manifold learning algorithms usually assume that the data lie on (or near) a manifold of intrinsic dimension d. In order to assess how well these algorithms perform under this hypothesis it is essential to draw data from a manifold where not only is this hypothesis known to be true but where the parameters – and all other unknown quantities – can be computed. This allows us to more precisely compare methods in otherwise optimal conditions. There

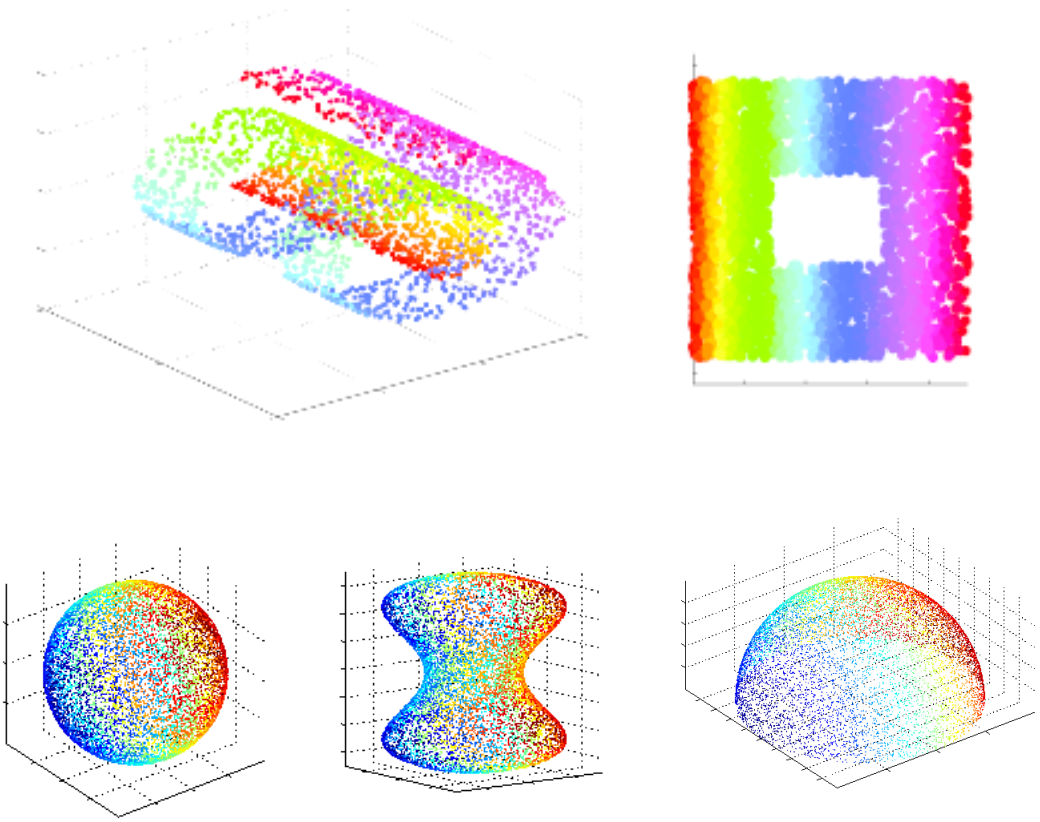

Figure 1.1: The figure on the top left displays a synthetic "swiss-roll with a hole" data set which is observed in $D = 3$ dimensions and then embedded into $d = 2$ dimensions, in this case a non-linear transformation can be performed to from 3 to 2 dimensions preserving all of the underlying distances and is an intuitively obvious one as seen on the top right. The bottom row displays the other three synthetic manifolds: sphere, hourglass, and dome. In all three cases the manifolds are of intrinsic dimension 2.

are a vast array of potential manifolds to chose from but we consider the following data sets due to their ubiquity in the manifold learning literature.

The first and one of the most common examples in the "Swiss-Roll with a Hole". The data set, which can be seen in 1.1 is a manifold with intrinsic dimension $d = 2$ as it is effectively a rectangle with a square hole cut out which has been loosely rolled in three dimensions. Points sampled from this object are in three dimensions however they clearly come from a two dimensional object – manifold. In this case it is also clear how to transform the data from three dimensions into two dimensions and preserve the original shape, as visualized on the right hand side of 1.1.

The second example we consider is a sphere and hourglass shape. Both of these data sets again appear in three dimensions but like the Swiss-Roll example they lie on manifolds of dimension $d = 2$. So just like the Swiss Hole data points sampled from these objects will be in three dimensions even though they lie on a manifold of dimension 2. Unlike the Swiss Roll it is not obvious (nor indeed possible) to transform into two dimensions without some sort of geometric distortion. This is, for example, why no two dimensional projection of the earth onto a single map is perfectly accurate. We also consider a dome which is also displayed in 1.1. Like the sphere the dome cannot be perfectly mapped into two dimensions without some sort of geometric distortion, however unlike the sphere it is possible to find a single chart (a homeomorphism) from the three dimensional data into two dimensions, however any such chart will still distort the geometry in some way (for example not all distances may be preserved).

1.2 Real World Data Sets

Manifold learning is ultimately interested in applying these algorithms to real world data sets where the manifold learning hypothesis is assumed (or proposed) to hold. Therefore in this section we describe a handful of real world data sets considered throughout this thesis as example applications of manifold learning.

One application occurs in astronomy. Due to the development of large-scale astronomical

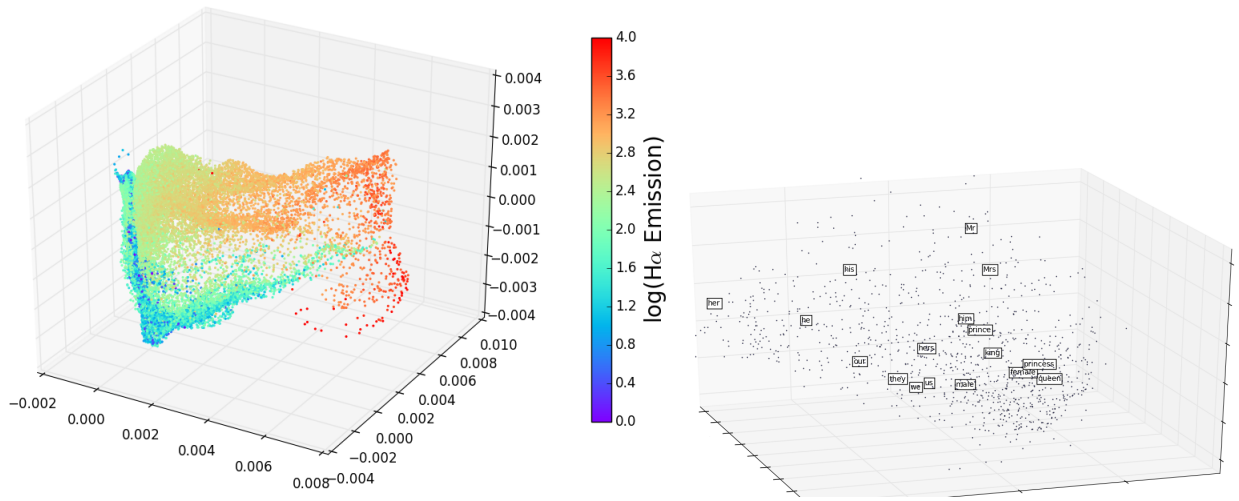

Figure 1.2: On the left we display an embedding (Laplacian Eigenmaps) of the the subset of the SDSS spectrum data consisting of 650 thousand galaxies in 3750 dimensions. The embedding is colored by Hydrogen Alpha emission which is a non-linear feature of the galaxies. On the right we display an embedding (Laplacian Eigenmaps) of a Word2Vec data set which contains 3,000,000 words and phrases initially embedded into 300 dimensions, here we display an embedding into 3 dimensions and some selected words which the points represent.

photometric and spectroscopic surveys such as the Sloan Digital Sky Survey (SDSS)[1] the amount of data available to astronomers is increasing significantly. Due to the large size of the data in both scale of samples and observed features (dimensions) it is becoming more important to identify and classify objects in a physically meaningful way. The Sloan Digital Sky survey captures the light spectrum emitted from a galaxy measured across a set of spectral bins. While a great deal of information is contained within the spectrum itself the rate at which it is sampled is arbitrary and large. Therefore, in order to provide a simple physical classification of the spectra, the data requires some form of dimension reduction. Principal Components Analysis (PCA), a linear dimension reduction technique, has been applied to galaxy spectra, but the linear nature of PCA prevents it from succinctly capturing non-linear features of the data such as dust obscuration or variation in spectral line widths. Specifically the variance preserving loss function of PCA prevents it from capturing high-frequency features which do not contribute significantly towards the total variance. It is therefore important to use a non-linear dimension reduction technique when working with these data. We refer the reader to [55] for a more detailed discussion of this application.

In this thesis we work with data taken from the Sloan Digital Sky Survey. In particular we consider a subset of the data from SDSS DR7 data release [2]. All data were shifted to a common rest-frame and cover a spectral range of 3830 Åto 9200 Ådivided into 3750 bins. The subsample used in this thesis consists of 675,000 galaxies which have sufficiently high signal to noise ratio called the "main sample;" taken from [49]. The data were pre-processed by first moving them to a common rest-frame wavelength and then filling-in missing data following [55] but using the more sophisticated weighted PCA algorithm of [16]. [1]

Recently the development of Word2Vec, an algorithm based on Neural Networks which are trained on a large corpus of text, can produce a numerical vector associated with each word in a given corpus. It was demonstrated [33] that some semantic content can be preserved when embedding a set of words and phrases in this manner although often into dimensions

[1]The data was provided to us after pre-processing by Jake VanderPlas, University of Washington eScience Institute.

between 100 to 1000. It is natural to ask whether or not a similar level of semantic content can be preserved with fewer dimensions and since the semantic content is being encoded via geometry (i.e. addition and subtraction of words) in particular we seek (non-linear) transformations of the Word2Vec data sets which also preserve geometry.

In this thesis we consider a Word2Vec data set published by Google based on Google News which consists of about 100 Billion words. The Model, a Continuous Bag-of-Words Neural Network, is trained to predict the current word from a window of surrounding words. The resulting embedding consists of 3 million words and phrases embedded into 300 dimensions. For further information on these data see [35]. We present an embedding into 3 dimensions using Laplacian Eigenmaps in figure1.2.

In chemistry there is an increasingly large amount of data from molecular dynamics (MD) simulations. There is a belief that the data from these simulations, while observed molecular configuration space is very large, the distribution of the physically relevant states is concentrated around a set of much lower dimension. This working assumption has been empirically verified by several works and is the motivation for applying non-linear dimension reduction to these molecular simulation data sets, in particular to find low dimensional coordinates that maintain physically important properties. We refer the reader to [44] for a more in-depth discussion of this application.

In figure 1.3 we produce embeddings (via Laplacian Eigenmaps) of some of these simulations. We use the output of a Molecular Dynamics Monte Carlo simulation for three different molecules: aspirin, ethanol, and malonaldehyde. For each molecule we observe the three dimensional coordinates of their atoms (for example 21 for apsirin). The observed coordinates are the output of a Metropolis Hastings sampling from a density defined based on energy. The data we consider is from [13][47] where more details about the simulation can be found [2]. Before embedding the data we first compute all of the pairwise distances between the atoms at each iteration and from that compute the angles created by the triangles created

[2]the data can be downloaded from: http://quantum-machine.org/datasets/

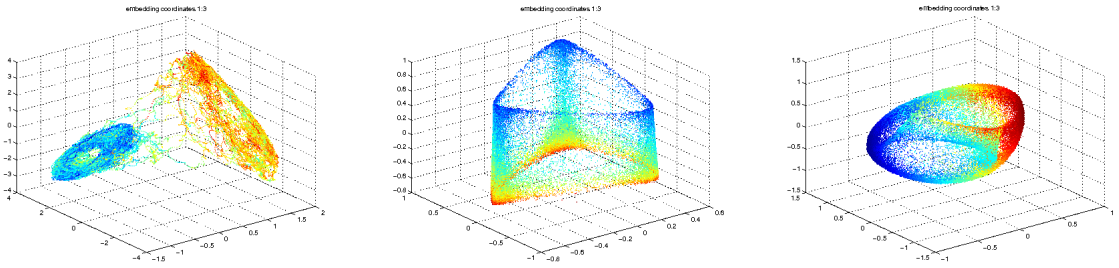

Figure 1.3: We display the first three coordinates of three different molecules analyzed in the Molecular Dynamics Monte Carlo simulation data. The molecules are aspirin, ethanol and malonaldehyde from left to right. Preliminary analysis of the aspirin molecule indicate that these embeddings have continuous preservation of torsion between atoms a feature of particular interest to chemists.

by all triples of the atoms. Angles that are above a specified threshold are excluded. From there a Singular Value Decomposition of the angles is taken and the leading s coordinates are taken such that the explained variance remaining is no more than 10^{-4}. For example for aspirin we select 50 coordinates so that the dimension of aspirin before embedding is 50. The data are sub-sampled (taking every k iterations) to a size of $50,000$. Although figure 1.3 displays all embeddings in three dimensions the actual embedding dimension was larger for all cases (dimension 9, 12, and 12 for aspirin, ethanol, and malonaldehyde respectively) and only the first three coordinates are displayed.

1.3 Thesis Outline

It is the overall goal of this thesis to provide researchers with the necessary tools to perform manifold learning on large scientific data sets. To this end we strive to provide in this thesis the background and definitions required for manifold learning, an understanding of existing algorithms as well as a novel algorithm, the necessary parameters required for using these algorithms as well as offering a cohesive method for selecting them, and finally an open

source software package to allow researches to easily apply these tools to large data sets.

In chapter 2 we cover the necessary background in differential geometry and how it relates to statistics that is required for the remainder of this thesis. In particular we discuss construction of the neighborhood graph which encodes the local structure of the data and is the primary way by which the data interacts with non-linear dimension reduction algorithms. We then introduce several of the algorithms which we will examine throughout this thesis. We introduce manifolds, Riemannian metrics, and isometry which gives us the vocabulary to discuss objects which have low intrinsic dimensionality and the ability to define transformations, isometries, which preserve geometry. Finally, to connect this to statistics we describe how to estimate Riemannian metrics from a data set. The work presented in this chapter is a summary of existing work and is intended as primer for the remainder of the thesis.

In chapter 3 we propose a novel algorithm `Riemannian Relaxation` for adjusting an embedding of a data set towards isometry into Riemannian space. We examine its performance on synthetic and real data sets. The primary novelty of this method is that it accepts both an embedding dimension s and a manifold dimension d. By accepting both `Riemannian Relaxation` is able to deal with the case when $d = s$ and when $d > s$ separately, this is the first algorithm to do so. This is important since it allows us to actively seek isometries to $\mathbb{R}^s$ when $s > d$ where the resulting embedding $\mathcal{N}$ is still a manifold of dimension d. We demonstrate the ability for `Riemannian Relaxation` to handle manifolds where $s > d$ as well as when $s = d$. Since `Riemannian Relaxation` is gradient based it also scales well with large data sets and we demonstrate this with an application in Astronomy.

In chapter 4 we discuss in more detail the methods for selecting parameters that were used throughout this thesis and that are offered by `megaman`. We discuss methods for estimating the bandwidth parameter which is required for constructing the neighborhood graph and dimension parameter which is required for the embedding. The individual methods presented in this chapter are not novel, however, they have been adjusted and presented coherently and sensibly. Furthermore, the Nyström method has been used as a method to scale eigendecompositions to large data sets we describe how this can also be used to solve

the out-of-sample extension problem for manifold learning in general.

In chapter 5 we briefly discuss how manifold learning as a subject requires large data sets in order to perform adequately. We present an open source python package `megaman` which aims to solve the problem of scaling manifold learning to large data sets. We use efficient algorithms for dealing with sparse matrices and solving the problems specific to manifold learning: finding nearest neighbor graphs and performing sparse eigendecompositions. We also provide the user with additional tools for estimating parameters and scaling spectral clustering.

In chapter 6 we discuss a related topic to manifold learning, spectral clustering. The operations on a Laplacian and eigendecomposition step relate these two methods and also allow for the tools of `megaman` to scale spectral clustering to large data sets. For very large graphs, however, it is essential that the graph undergo some form of pruning before pursuing spectral clustering. Therefore we also provide a method, `PruneGraphForClustering`, which is a fast, low memory, algorithm which we use to prune the graph before the embedding step in a clustering algorithm. We motivate this section with an example using a citation network from Jstor.

Chapter 2

BACKGROUND

2.1 Non-linear Dimension Reduction

Non-linear dimension reduction is an *unsupervised* learning technique which accepts as input a data set $\mathbf{X} \in \mathbb{R}^{n \times D}$ and outputs an new data set $\mathbf{Y} \in \mathbb{R}^{n \times s}$ where here n denotes the number of observed data points and D is the dimension of each observed data point (i.e. the number of features per data point). Non-linear dimension reduction applies a (usually implicit) non-linear function ϕ to the rows of $\mathbf{X}$ to produce the rows of $\mathbf{Y}$. In order for non-linear dimension reduction to be dimension reduction we have that $D > s$ and usually D is significantly greater than s (often an order of magnitude or more) in applications. In principle non-linear dimension reduction requires only a data set $\mathbf{X}$, however, in practice these algorithms usually require a set of parameters. In particular, non-linear dimension reduction requires a dimension s (which will be the dimension of the points *output* by the non-linear dimension reduction algorithm) as well as a *bandwidth* parameter (which will be defined more precisely later) which determines the neighborhood structure of the data which we desire to preserve.

2.2 Differential Geometry

In order to explain the contents of this thesis a sufficient background is required. In the later chapters we use the terminology, definitions, and theorems of Riemannian Manifolds to motivate our methods for non-linear dimension reduction and parameter selection. This section defines the necessary concepts in differential geometry that we will use in this thesis. We also explain how we relate differential geometry where the entire manifold is observed to statistics where we only observe a random selection of points on the manifold. The contents of this section are presented here as necessary background information, for a more complete overview of differential geometry we refer the reader to [27] and for a complete discussion of estimating Riemannian metrics from a data set we refer the reader to [42].

To formalize the notion of an object which has an ambient dimension D and an intrinsic dimension d we need the language of *smooth manifolds*. In addition we are interested in

manifolds which admit transformations which preserve geometry, therefore we need to introduce the notion of a *Riemannian Metric* and an associated Riemannian-Manifold. Equipped with this we can define an *isometric transformation*.

To understand these notions we must define a manifold, charts, atlases and finally a smooth manifold.

Definition 1. *(Manifold) A $d-$dimensional* ***manifold*** $\mathcal{M}$ *is a topological space such that every point is homeomorphic to an open set of* $\mathbb{R}^d$.

Definition 2. *(Chart and Atlas) A chart (U, x) of a manifold $\mathcal{M}$ is an open set U along with a homeomorphism $x : U \to V \subset \mathbb{R}^d$ where V is an open subset. A C^∞ atlas $\mathcal{A}$ is a collection of charts:*

$$\mathcal{A} \equiv \bigcup_{i \in I} (U_i, x_i)$$

With I and index set such that the atlas covers the manifold $\mathcal{M}$:

$$\mathcal{M} = \bigcup_{i \in I} U_i$$

and does so in a smooth way, that is for any $i, j \in I$ the transition map:

$$x_i \circ x_j^{-1} : x_j \left(U_i \cap U_j \right) \to \mathbb{R}^d$$

is continuous and infinitely differentiable.

Definition 3. *(Smooth Manifold) A* ***smooth manifold*** $\mathcal{M}$ *is a manifold with a C^∞-Atlas $\mathcal{A}$*

Thus a smooth manifold is a topological space that, locally, can be mapped into $\mathbb{R}^d$ homeomorphically, that is with a continuous function x_i that has continuous inverse. That is to say, locally around each point the manifold $\mathcal{M}$ "*looks like*" $\mathbb{R}^d$. Although nothing in this definition requires that the manifold $\mathcal{M}$ be observed in any Euclidean space, in the case of *manifold learning* we restrict ourselves to cases where $\mathcal{M} \subset \mathbb{R}^D$ for some $D \gg d$, that is $\mathcal{M}$ is a *sub-manifold* of $\mathbb{R}^D$.

2.2.1 Embeddings

Armed with the notion of a smooth manifold we can define one goal of *manifold learning*. Suppose we observe data points $\mathcal{D} = \{x_1, \ldots, x_n\} \subset \mathcal{M}$ sampled from a smooth manifold $\mathcal{M}$ with intrinsic dimension d which is itself a submanifold of D-dimensional Euclidean space $\mathcal{M} \subset \mathbb{R}^D$. The task of manifold learning is to provide a non-linear, continuous, and smooth mapping $\phi : \mathcal{M} \to \mathcal{N}$ (where $\mathcal{N} \subset \mathbb{R}^s$) of the manifold $\mathcal{M}$ into lower dimensional space $s \ll D$, in particular we seek an *embedding* of the data $\mathbf{X}$.

Definition 4. *(Embedding) Let $\mathcal{M}$ and $\mathcal{N}$ be smooth manifolds and let $\phi : \mathcal{M} \to \mathcal{N}$ be a smooth, injective, map with $rank(f) = dim(\mathcal{M})$, then f is called an* immersion*, if additionally f is a homeomorphism onto $f(\mathcal{M})$, then f is an* embedding*.*

The rank of a map $\phi : \mathcal{M} \to \mathcal{N}$ is equal to d if the Jacobian $\mathcal{T}_p\mathcal{M} \to \mathcal{T}_{\phi(p)}\mathcal{N}$ of the map ϕ has rank d for all points $p \in \mathcal{M}$. And $\mathcal{T}_p\mathcal{M}$ denotes the *tangent plane* of $\mathcal{M}$ at the point p. The question becomes: when can such a map ϕ be found, and when does such a map ϕ *preserve the geometry of* $\mathcal{M}$?

First it is obvious that a map ϕ into $\mathbb{R}^d$ is not always possible, for example a 2-sphere $S^2 \subset \mathbb{R}^3$ cannot be embedded into $\mathbb{R}^2$ using a single map U, that is we can't restrict ourselves to looking for a single *global map* U. If we allow $s \geq d$ then we may look for a single map ϕ. For example the sphere can be naturally embedded into $\mathbb{R}^3$ (even though it has intrinsic dimension $d = 2$).

According to the Whitney Embedding Theorem [27] we know that $\mathcal{M}$ can be embedded smoothly into $\mathbb{R}^{2d}$ using one homeomorphism ϕ. It is known that for some manifolds this bound is tight, that is there can be no embedding into $\mathbb{R}^s$ with $s \leq 2d$, however, for many manifolds in practice this may be overly-pessimistic. Hence we seek one smooth map $\phi : \mathcal{M} \to \mathbb{R}^s$ with $d \leq s \leq 2d \ll D$. If $s \ll D$ then a significant *dimension reduction* can be achieved.

Smooth embeddings preserve the topology of the original $\mathcal{M}$. Nevertheless, in general, they distort the geometry. What we mean here is that an embedding $\phi : \mathcal{M} \to \mathbb{R}^s$ may be one

that preserves neighborhoods (that is, neighborhoods in the high dimensional ambient space are mapped into neighborhoods of the low dimension embedding space) but the question is does it also preserve distances? Or additionally does it preserve volumes, arc-lengths, areas, etc? Usually although neighborhoods are preserved the area around them is *stretched* in some way such that distances computed on the embedding manifold will be stretched versions of those computed in the ambient space. The question is then can we find an embedding ϕ which minimizes, or possibly eliminates, this stretching? I.e. an embedding which *preserves the geometry* of $\mathcal{M}$? Theoretically speaking[1], preserving the geometry of an embedding is embodied in the concepts of *Riemannian metric* and *isometric embedding*.

2.2.2 Geometry

Recall that in standard *Euclidean* geometry all geometric quantities, such as distances, areas, volumes, etc., are encoded using an *inner product* between two vectors in $\mathbb{R}^d$: $\langle u, v, \rangle = \sum_{i=1}^{d} u_i v_i$. Armed with an inner product we can compute any geometric quantity we are interested in for example the angle between two vectors follows from: $\langle u, v \rangle = ||u|| \cdot ||v|| \cos(\theta)$. Similar to Euclidean spaces a Riemannian Manifold has an operator which defines an inner product on the tangent subspace called a Riemannian metric.

Definition 5. *(Riemannian Metric) a Riemannian Metric g is a symmetric positive definite tensor field on $\mathcal{M}$ which defines an inner product $\langle , \rangle_g$ on the tangent space $\mathcal{T}_p\mathcal{M}$ for every point $p \in \mathcal{M}$.*

So that

Definition 6. *(Riemannian Manifold) A* Riemannian manifold *is a smooth manifold $\mathcal{M}$ with a Riemannian metric g which defines an inner product on the tangent subspace at every point. A Riemannian Manifold $\mathcal{M}$ equipped with a Riemannian Metric is denoted as $(\mathcal{M}, g)$.*

[1]For a more complete presentation the reader is referred to [21] or [42] or [26].

Just like the standard Euclidean inner product is used to define geometric quantities in Euclidean spaces the Riemannian metric defines geometric quantities on Riemannian Manifolds. The inner product between two vectors u, v on tangent plane at point p:

$$\langle u, v \rangle_g = \sum_{i=1}^{d} \sum_{j=1}^{d} g_{ij} u_i v_j \tag{2.1}$$

Then we define quantities analogously to the Euclidean case, for example the norm of a vector is $||u||_g = \sqrt{\langle u, u \rangle_g}$ similarly for a curve c parameterized by $t \in \mathbb{R}$, by $c : [a, b] \to \mathcal{M}$ has length:

$$l(c) = \int_a^b \sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} g_{ij} \frac{dx^i}{dt} \frac{dx^j}{dt}} dt$$

with $(x^1, \ldots, x^d)$ coordinates of a chart (U, x) and a curve inside the chart $c \subset U$. This can also be extended to overlapping charts via the transition map. See [26] for an overview of calculus on Manifolds.

Next we move to maps ϕ from one Riemannian Manifold $\mathcal{M}$ to another Riemannian Manifold $\mathcal{N}$ which are geometry preserving. These maps are called *isometries* and, intuitively, are maps which *preserve the Riemannian Metric.* More formally

Definition 7. *(Isometry) A diffeomorphism $\phi : \mathcal{M} \to \mathcal{N}$ is called an isometry iff for all $p \in \mathcal{M}, u, v \in \mathcal{T}_p\mathcal{M}$ we have $< u, v >_{g_p} = < d\phi_p u, d\phi_p v >_{h_{\phi(p)}}$. Where here $d\phi_p$ denotes the Jacobian of ϕ at the point p on $\mathcal{M}$, that is, the map $d\phi : \mathcal{T}_p\mathcal{M} \to \mathcal{T}_{\phi(p)}\mathcal{N}$.*

An isometry refers to a diffeomorphism ϕ that is applies to $\mathcal{M}$ and maps it into $\mathcal{N}$. When we refer to the embedding itself $\phi(\mathcal{M})$ then we can define an *isometric embedding* as follows:

Definition 8. *(Isometric Embedding) an embedding $\phi(\mathcal{M})$ where $\phi : \mathcal{M} \to \mathcal{N}$ of a manifold $\mathcal{M}$ is isometric if $(\phi(\mathcal{M}), h \mid_{f(\mathcal{M})})$ is isometric to $(\mathcal{M}, g)$ where $h \mid_{f(\mathcal{M})}$ is the restriction of h, the Riemannian metric of the manifold $\mathcal{N}$, such that $\phi(\mathcal{M}) \subset \mathcal{N}$, to tangent space $\mathcal{T}_{f(p)}\phi(\mathcal{M})$.*

Since the Riemannian Metric is preserved under isometries this implies that all geometric quantities, such as path lengths, areas, volumes, angles, etc are preserved. It makes sense, therefore, to define isometric embeddings as our notion of "geometry preserving" embeddings.

When we take a Riemannian manifold $(\mathcal{M}, g)$ and apply a function $\phi : \mathcal{M} \to \mathcal{N}$ to the manifold the function ϕ induces a metric h, called a pushforward metric which is defined such that (M, g) and (N, h) are isometric. The pushforward metric is defined as follows:

Definition 9. *(Pushforward Metric) If ϕ is an embedding of a Riemannian manifold $(\mathcal{M}, g)$ into another manifold $\mathcal{N}$, then the* pushforward metric *h of the metric g induced by ϕ is defined as:*

$$\langle u, v\rangle_h = \langle d\phi_p^{-1}u, d\phi_p^{-1}v\rangle_{g_p}$$

That is, to compute the inner product in the tangent space of $\mathcal{N}$ you invert the tangent map to $\mathcal{T}_p\mathcal{M}$ and then compute the inner product using g. Therefore $(\mathcal{M}, g)$ and $(\phi(\mathcal{M}), h)$ are isometric by construction.

The final important question that we address is whether a submanifold $\mathcal{M} \subset \mathbb{R}^D$ can be embedded isometrically into a *Euclidean* space. By embedding into Euclidean space we mean that the inner product on the tangent plane is simply the inner product induced by the standard dot product of the space $\mathbb{R}^s$. If we denote this as δ_s then we want a map ϕ such that $(\mathcal{M}, \delta_D)$ is isometric to $(\phi(M), \delta_s)$. This was answered by John Nash in the following theorem:

Theorem 10. *Nash's Embedding Theorem If $\mathcal{M}$ is a $d-$dimensional manifold of class C^k for $3 \leq k \leq \infty$ then there exists a number $s \leq d(3d+11)/2$ if $\mathcal{M}$ is compact and $s \leq d(d+1)(3d+11)/2$ if $\mathcal{M}$ is not compact, and an injective map $\phi : \mathcal{M} \to \mathbb{R}^s$ also of class C^k such that for $u, v \in \mathcal{T}_p\mathcal{M}$*

$$\langle u, v\rangle = \langle d\phi_p(u), d\phi_p(v)\rangle$$

i.e. $(\mathcal{M}, \delta_D)$ is isometric to $(\phi(\mathcal{M}), \delta_s)$

Therefore we are interested in isometric embeddings into Euclidean spaces. We are interested in this primarily so that we do not have to carry around a pushforward Riemannian

Metric in order for our embedding to remain isometric. We note that the method developed by Nash in his proof is not particularly amenable to an algorithm for finding ϕ.

Finally since we are dealing directly with the embedding $\phi(\mathcal{M}) \subset \mathbb{R}^s$ where $s \leq d$ we have to deal with the rank-deficient pushforward Riemannian Metric on the embedding rather than on the tangent plane. We follow [42] who define the following Embedding pushforward Riemannian Metric:

Definition 11. *Embedding (Pushforward) Metric Let* $\phi : \mathcal{M} \to \mathbb{R}^s$
then for $u, v \in \mathcal{T}_{\phi(p)}\phi(\mathcal{M}) \bigoplus \mathcal{T}_{\phi(p)}\phi(\mathcal{M})^{\perp}$

$$\langle u, v \rangle_{h(\phi(p)} \equiv \langle d\phi_p^{\dagger}(u), d\phi_p^{\dagger}(v) \rangle_{g(p)} \tag{2.2}$$

where $d\phi^{\dagger}$ *is the Moore-Penrose pseudoinverse of the Jacobian of* ϕ.

This ensures that $(\mathcal{M}, g)$ and $(\phi(\mathcal{M}), h)$ are isometric. Therefore, one goal of Manifold Learning as a field of machine learning can be states as: finding an isometry $\phi : \mathcal{M} \to \mathbb{R}^s$ given a sample $\mathcal{D}$ from $\mathcal{M}$.

2.3 Manifold Learning Algorithms

Since we will be working with finite data sets we require a brief comment on notation. When we describe algorithms acting on data, we will use coordinate and finite sample representations. The data is $\mathbf{X} \in \mathbb{R}^{n \times D}$, and an embedding thereof is denoted $\mathbf{Y} \in \mathbb{R}^{n \times s}$. In some equations we find it important to distinguish between rows and columns of a matrix. We denote rows k of $\mathbf{X}, \mathbf{Y}$, by $\mathbf{X}_{k:}$, $\mathbf{Y}_{k:}$ and are coordinates of data point k, while the columns, e.g $\mathbf{Y}_{:j}$ represent functions of the points, i.e restrictions to the data of functions on $\mathcal{M}$. Occasionally we also somewhat overload this notation particularly with the estimated Riemannian metric at a point i will be denoted $\mathbf{H_i}$ since this will be used mostly within a sum over data points the notation should be clear. Finally we will denote the $i-j$th component of a matrix by $\mathbf{W}_{ij}$.

Most Manifold Learning algorithms rely on constructing a *neighborhood graph* $\mathcal{G} = (V, E)$ using the data $\mathbf{X}$ where the vertices (nodes) in V correspond to the data points $1, \ldots, n$. An

edge between node i and j is included in the graph if point $\mathbf{X}_{i:}$ and $\mathbf{X}_{j:}$ are "neighbors" where neighbors is defined either by a radius parameter or a limit on the number of neighbors. We describe three different methods for constructing such a graph shortly but in general the edges between two nodes encodes some sense of local geometry or similarity. Usually the graph is constructed so that it is sparse, in particular there is an edge between two data points i and j if the are neighbors where we don't allow for the graph to be fully connected. It is often the case that the set of neighbors to the point i, i.e. the set of points j such that there is an edge between i and j in the graph, is used to estimate the tangent space at the ith point. Therefore, it is through these neighbors that we define the "local" structure of the data, and in particular, the structure which we wish to preserve during the non-linear transformation. Points that are close together in the observed data ought to remain close together in the embedded data. There are several types of neighborhood graphs: the k-nearest-neighbors graph, the radius-neighbors graph and the truncated Gaussian kernel graph.

The first consists of creating a k-nearest neighbors graph where the edge between point i and j is included if $\mathbf{X}_{j:}$ is one of the k nearest neighbors of $\mathbf{X}_{i:}$ where we use the Euclidean distance between points as our metric for nearness: $d_{ij} = ||\mathbf{X}_{i:} - \mathbf{X}_{j:}||$. If I_k is the set of indices to points such that $|I_k| = k$ and $\forall j \in I_k \; d_{ij} < d_{il}$ for $l \notin I_k$. This graph is a weighted graph with the edge weight between i and j to be d_{ij} whenever $(i, j) \in E$. The interpretation of the parameter k is fairly simple as it takes on an integer value. In general the Euclidean distances d_{ij} that are included in the neighborhood graph are meant to be good approximations to the distance on the manifold between point i and point j as included points are supposed to lie near the tangent plane and so the Euclidean distance between the two points approximates the distance on the tangent space. If points are sampled non-uniformly then a parameter k may select too many neighbors for a point where the data is sampled sparsely making the interpretation of the k parameter challenging geometrically.

The second method constructs a graph which includes $(i, j) \in E$ if $d_{ij} < r$, that is if the distance is smaller than a prescribed radius. This is similar to the k-nearest neighbors graph wherein only "near" points are included but the difference is that nearness is defined by a

maximum distance rather than a maximum size of the set of near points. Consequently some points may have a larger set of edges connected to them than others. The interpretation of the radius parameter is geometrically motivated as it defines a ball around each data point as the neighborhood. Selecting the parameter r can be challenging, selecting it too small can make the graph disconnected such that creating an embedding that connects all of the data points challenging. If the parameter is selected to be too large then the distances fail to capture the local properties of the data and the euclidean distance may no longer be an appropriate estimate of the true distance between the two points. The notion of euclidean distance versus true distance will be elaborated upon in the following section. In general we take the weight of the edge again to be d_ij.

The third method is to use a weighted graph where the weights are constructed by applying the Guassian kernel to the distances: $w_{ij} = \exp(-d_{ij}^2/2h^2)$. In this case the graph is fully connected but has weights that decay as the distance between the two points increase. Where we called r a radius parameter when it is used as a cut-off between including and excluding edges we will call h a "bandwidth" parameter as it is used in a Gaussian Kernel. Like the radius-neighbors graph this Kernel graph is geometrically motivated but produces a dense graph. In practice, however, it is often truncated such edges (i, j) are included in E only if $d_{ij}^2 < rh$ for some constant r. This re-introduces the issue with the previous two methods in that selecting a bandwidth (or constant) too small can lead to disconnected graphs. In general for the Gaussian Kernel it's common to use $r = 3$ since 99.9% of the mass of a Standard Gaussian random variable lies within 3 standard deviations of 0.

In all three cases we can take the graph and encode it as an *adjacency* matrix $\mathbf{A}$ which is $n \times n$ and has entries $A_{ij} = d_{ij}$ in the first two cases and $A_{ij} = w_{ij}$ in the third case. In particular for the third case we get a (sparse) weight matrix:

$$[\mathbf{W}]_{ij} = \begin{cases} \exp\left(-\frac{||\mathbf{X}_i - \mathbf{X}_j||^2}{2h^2}\right) & \text{if } ||\mathbf{X}_i - \mathbf{X}_j|| \leq 3h \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

For the k-nearest neighbors and $r-$radius neighbors graph these Adjacency matrices are

distance matrices – small weights indicate closeness – and the omitted edges ought to be thought of as "infinite" rather than zero. On the other hand, for the truncated Guassian matrix, the edges encode *similarity* between vertices (where $w_{ij} = 1 \Leftrightarrow \mathbf{X}_i = \mathbf{X}_j$) whereas in the previous methods edge weight encodes distance (dissimilarity). Consequently, this method has the advantage that edges not included in the graph have the natural interpretation of being zero rather than infinite.

2.3.1 Existing Methods

There are many existing non-linear dimension reduction algorithms, too many to list them exhaustively. Several algorithms have direct motivation by manifolds (discussed in the next section) and to which we compare the method of chapter 3. We briefly describe the methodologies below. We note that all of them involve one of the neighborhood graphs as described in the previous section this is a common theme between non-linear dimension reduction algorithms and one which we exploit in chapter 5.

- **Multidimensional Scaling** [8](MDS): MDS is not a non-linear dimension reduction algorithm, however, it is directly applied by Isomap and so we describe it for completeness. MDS is a spectral method that begins with a matrix of pairwise dissimilarities (potentially distances) and finds an embedding which attempts to preserve these dissimilarities as euclidean distances. It works by exploiting the relationship between the distance matrix and $\mathbf{X}\mathbf{X}'$.

- **Isomap** [51]: the goal of Isomap is to find a distance preserving transformation of the data. It does this in two stages. First it estimates all of the true distances by using the k-nearest neighbors or radius neighbors graph and applying the graph-shortest-path algorithm to estimate *all* of the pairwise distances. Next Isomap applies MDS to this new distance matrix to create an embedding.

- **Laplacian Eigenmaps** (LE) [5]: LE is another spectral method which exploits the

relationship between the graph Laplacian as described in the next section and smooth embeddings. In particular LE takes the s eigenvectors of the graph Laplacian corresponding to the s smallest eigenvalues (excluding zero) which has been shown to preserve local distances with a smooth embedding.

- **Hessian Locally Linear Embedding** [17] (HLLE): Similar to Laplacian Eigenmaps, this method is also a spectral method which begins with constructing the neighborhood graph. Each neighborhood is used to construct local tangent coordinates which is then used to construct local Hessian estimates and generate a quadratic form. Next an eigendecomposition of the Hessian quadratic form is computed and the smallest non-constant eigenvectors are taken as the embedding.

- **Maximum Variance Unfolding** [56] (MVU): Similar to other methods MVU optimizes a function which encodes a distance preserving relationship. Again a neighborhood graph is constructed and the constraint of the optimization is that all distances in this neighborhood graph be preserved in the embedding and otherwise all remaining distances be maximized (hence unfolding). The resulting constrained optimization can be solved as a semi-definite program.

One other algorithm that is of interest that, unlike the above, optimizes a cost function by gradient descent is t-SNE [53]. t-SNE uses the weighted graph using the Guassian Kernel to compute a probability distribution across the pairs of data points. For a given embedding $\mathbf{Y}$ a similar, but not identical, graph is computed and a distribution across pairs of data points in the embedding is computed. The object of t-SNE is to minimize the KL divergence between these two distributions and it does so by gradient descent. It is noted by the authors that t-SNE is designed to embed into small dimension $d = 2, 3$ explicitly for the task of visualization, and does not take into account an attempt to preserve geometry between the two embeddings.

2.4 Estimating Riemannian Metrics

To reconstruct the Embedding Metric from data we exploit the relationship between the Laplace-Beltrami operator and the Riemannian metric on a manifold [27].

Definition 12. *(Laplace-Beltrami Operator) The Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ is an operator on functions f on a manifold $\mathcal{M}$ and is defined as the divergence of the gradient:*

$$\Delta_{\mathcal{M}} f = \nabla^2 f = \nabla \cdot \nabla f \tag{2.4}$$

In particular if (U, x) is a local chart of $\mathcal{M}$ and if $\partial_i = \frac{\partial}{\partial x^i}$ the basis vectors, with g the Riemannian metric g, then from [45] we have that:

$$\Delta_{\mathcal{M}} f = \frac{1}{\sqrt{|g|}} \sum_{ij} \partial_i \left(\sqrt{|g|} g^{ij} \partial_j f \right)$$

Where $|g|$ denotes the determinant and g^{ij} denotes the components of the inverse of g.

Therefore given $\Delta_{\mathcal{M}}$ we can recover g as follows: If $\Delta_{\mathcal{M}}$ is the Laplace-Beltrami operator then the co-metric at a point p can be written as:

$$h^{ij} = \Delta_{\mathcal{M}} \frac{1}{2} (f^i - f^i(p))(f^i - f^i(p))|_{f^i = f^i(p), f^j = f^j(p)} \tag{2.5}$$

Where here h denotes the pseudo-inverse of g. We next describe how [42] use this to estimate the Riemannian metric using a data set of points sampled from the manifold $\mathcal{M}$.

Based on this, [42] gives a a construction method for a discrete estimator of the Riemannian metric g, in any given coordinate system, from an estimate $\boldsymbol{\mathcal{L}}$ of $\Delta_{\mathcal{M}}$ by way of 2.5.

The construction of the "graph-Laplacian" $\boldsymbol{\mathcal{L}}$ requires a *kernel*, which can be the (truncated) Gaussian kernel $K_h(z) = \exp(-z^2/2h^2)$, $|z| < rh$ for some fixed $r > 0$ [22, 52]. To construct $\boldsymbol{\mathcal{L}}$ we use the method of [14], which guarantees that, if the data are sampled from a manifold $\mathcal{M}$, $\boldsymbol{\mathcal{L}}$ converges to $\Delta_{\mathcal{M}}$, the Laplace-Beltrami Operator on a manifold (to be defined later) [22, 52]. In order to estimate the Embedding Metric we will need to estimate $\boldsymbol{\mathcal{L}}$ which requires the (truncated) Gaussian kernel weighted radius neighbors graph of the

previous section in particular denote it's adjacency matrix by $\mathbf{W}$ as in 2.3. Next, construct $\boldsymbol{\mathcal{L}} = [\boldsymbol{\mathcal{L}}_{kl}]_{ij}$ of $\mathcal{G}$ as follows. First, symmetrically normalize $\mathbf{W}$, let $\mathbf{D}$ be the diagonal matrix with $D_{ii} = \sum_{j=1}^{n} W_{ij}$ and $D_{ij} = 0$ otherwise, or $\mathbf{D} = \text{diag}(\mathbf{W1})$ for the one-vector $\mathbf{1}$. Then let

$$\tilde{\mathbf{W}} = \mathbf{D}^{-1}\mathbf{W}\mathbf{D}^{-1}$$

Again find the diagonal matrix of the new weight matrix: $\tilde{\mathbf{D}} = \text{diag}(\tilde{\mathbf{W}}\mathbf{1})$ Then finally:

$$\boldsymbol{\mathcal{L}} = (\epsilon h^2)^{-1}(\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{W}} - \mathbf{I}_n) \tag{2.6}$$

Where ϵ depends on the kernel, $K_h(\cdot)$ but for the Guassian Kernel $\epsilon = 1/4$. Equation (2.4) represents the discrete versions of the renormalized Laplacian construction from [14]. Note that $\mathbf{W}, \mathbf{D}, \tilde{\mathbf{D}}, \tilde{\mathbf{W}}, \boldsymbol{\mathcal{L}}$ all depend on the bandwidth h via the heat kernel. The consistency of $\boldsymbol{\mathcal{L}}$ for $\Delta_{\mathcal{M}}$ has been proved in [22, 52].

In a given coordinate representation $\mathbf{Y}$, a Riemannian metric g at each point is an $s \times s$ positive semi-definite matrix of rank d. The method of [42] obtains the matrix Moore-Penrose pseudoinverse of this metric (which must be therefore inverted to obtain the pushforward metric). We denote this inverse at point k by $\mathbf{H}_k$; let $\mathbf{H} = [\mathbf{H}_k,\, k = 1, \ldots n]$ be the three dimensional array containing the inverse for each data point. Note that $\mathbf{H}$ is itself the (discrete estimate of) a Riemannian metric, called the (pushforward) co-metric.

Let $\boldsymbol{\mathcal{L}}$ be the graph Laplacian of the data points $\mathbf{X} \in \mathbb{R}^{n \times D}$ computed using and let let $\mathbf{Y} \in \mathbb{R}^{n \times s}$ be the embedded points. Given the set of points, $\mathbf{Y}$, and Graph Laplacian, $\boldsymbol{\mathcal{L}}$, equation (2.5) becomes:

$$\mathbf{H}^{ij} \;=\; \frac{1}{2}\left[\boldsymbol{\mathcal{L}}(\mathbf{Y}_{:i} \cdot \mathbf{Y}_{:j}) - \mathbf{Y}_{:i} \cdot (\boldsymbol{\mathcal{L}}\mathbf{Y}_{:j}) - \mathbf{Y}_{:j} \cdot (\boldsymbol{\mathcal{L}}\mathbf{Y}_{:i})\right] \tag{2.7}$$

Where here $\mathbf{H}^{ij}$ is the vector whose kth entry is the ijth element of the pushforward co-metric $\mathbf{H}$ at the point k and $\cdot$ denotes element-by-element multiplication. We later show that $\mathbf{H}_k$ itself, i.e. the $s \times s$ matrix that estimates the pushforward co-metric at point $\mathbf{Y}_{k:}$ can be written as a quadratic form with respect to $\mathbf{Y}$ and a positive semi-definite matrix. To summarize, given a data set $\mathbf{X}$ we compute the graph Laplacian $\boldsymbol{\mathcal{L}}$ then given an embedding

$\mathbf{Y}$ we compute the R metric at each point $\mathbf{H}_k$ using using 2.7. In chapter chapter 3 we discuss how to update the embedding $\mathbf{Y}$ to optimize for isometry at each point by minimizing a loss function with respect to $\mathbf{H}$.

In figure 2.1 we demonstrate how the estimate of a Riemannian metric acquired in this manner can be visualized for 2 dimensional embeddings. We show how data lying on a circle that is then "stretched" into an ellipse causes the (Riemannian co-metrics which would other wise be circles to appear as ovals.) We display what is called the "co-metric" rather than the Riemannian Metric itself. The co-metric is the (pseudo)-inverse of the pushforward metric as in (2.5), Where the pushforward metric gives a scale to *correct* the embedding in order to compute the inner product the co-metric displays the "stretch" or distortion induced by the embedding. In the simple case where $d = 1$ the metric tensor g is a scalar such that the dot product between u and v is computed as $\langle u, v \rangle = guv$ so that if $g > 1$ then the dot product $u \cdot v$ is *too small* (so that here g large implies distances are too small) on the other hand if we let $h = 1/g$ be the co-metric then whenever h is large the distance in the embedding is too large and when h is small the distance in the embedding is too small. The the co-metric, when plotted, then indicates the direction and degree that stretching occurred when mapping from the original manifold $\mathcal{M}$ to $\mathcal{N}$ via ϕ rather than the degree by which the embedding ought to be corrected. This makes viewing the co-metric h more visually intuitive as displayed in 2.1.

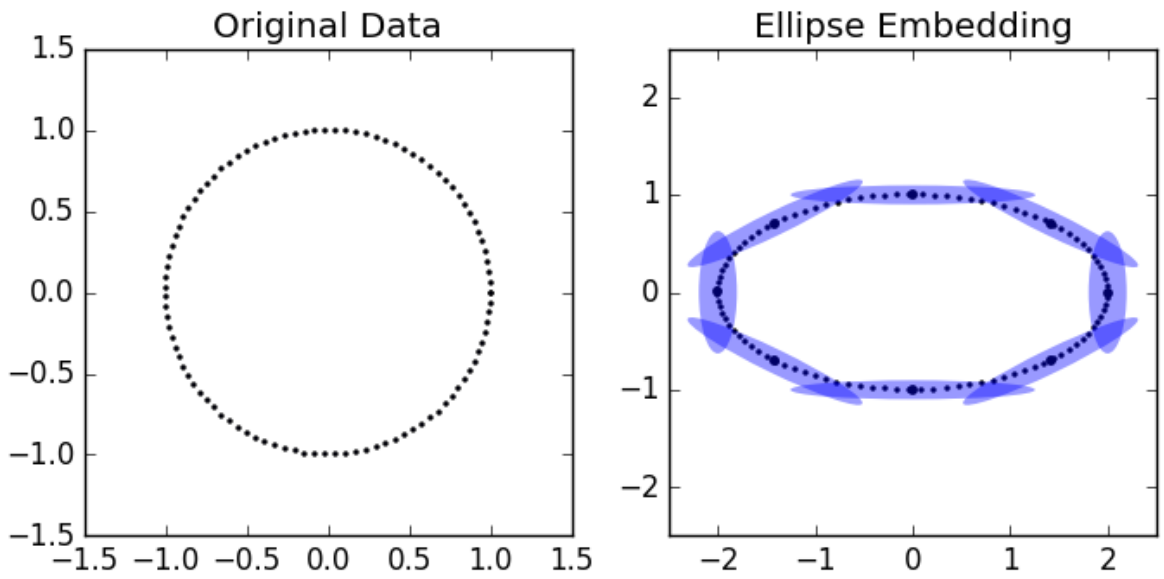

Figure 2.1: The circle on the left hand plot represents the original data with $D = 2$. The ellipse in the right hand plot is an "embedding" of these data into $s = 2$ dimensions, which deforms the circle into an ellipse. The blue ovals plotted at selected points represent the Riemannian co-metric at that point. The long axes of the ellipses represent the local unit of length along the ellipse (note: the short axes are artifacts); the units are smaller left and right, where the length of the original curve was compressed, and larger in the middle where the circle was stretched.

Chapter 3

GEOMETRY PRESERVING NON-LINEAR DIMENSION REDUCTION

3.1 Overview of Riemannian Relaxation

In this chapter we describe an algorithm, Riemannian Relaxation, which attempts to find an isometric embedding $\phi(\mathbf{X})$ into $\mathbb{R}^s$ given an data set $\mathbf{X}$ assumed to be sampled from a d dimensional Riemannian sub-manifold of $\mathbb{R}^D$. Similar to the methods as described in the previous section Riemannian Relaxation seeks an en embedding which minimizes a loss function. The main difference, from the algorithmic point of view, is that the loss function we propose does not have a form amenable to a standard solver (and is not even guaranteed to be convex or unimodal). Thus, we do not obtain a mapping ϕ in "one shot", as the previous algorithms do, but by the gradual improvements of an initial guess, i.e. by gradient descent.

The algorithm is initialized with a smooth embedding $\mathbf{Y} = \phi(\mathbf{X}) \subseteq \mathbb{R}^s$, $s \geq d$; we define the objective function Loss($\mathcal{Y}$) as the averaged deviation of the pushforward metric from isometry. Then $\mathcal{Y}$ is iteratively changed in a direction that decreases Loss. The construction of the loss is based on two main sets of results. First, an estimator $\boldsymbol{\mathcal{L}}$ of the *Laplace-Beltrami* operator $\Delta_{\mathcal{M}}$ of $\mathcal{M}$, and second, an estimator of the pushforward metric g in the current coordinates $\mathbf{Y}$ as described in §2.4.

3.2 Defining a Loss Function

We aim to derive a loss function which, using the estimated pushforward Riemannian Metric, encodes deviation from isometry. In the previous chapter we demonstrated that a pushforward metric automatically implies isometry, however, we specifically seek an isometry into *Euclidean* space $\mathbb{R}^s$, that is our pushforward metric must take on a specific form, i.e. the pushforward metric $h = \delta_s$. When the data $\mathbf{X}$ is sampled from a manifold $\mathcal{M} \subset \mathbb{R}^D$ of dimension d which we embed into $\mathbb{R}^s$ there are two cases which we must consider separately: when $s = d$, i.e. there exists an isometry $\phi : \mathcal{M} \to \mathbb{R}^d$, and when $s > d$, that is in order for $\phi : \mathcal{M} \to \mathbb{R}^s$ to be an isometry we require that $s > d$. The first case is simpler but much more restrictive, nevertheless it is important to examine it first as it will inform the

Input : data $\mathbf{X} \in \mathbb{R}^{n \times D}$, kernel function $K_h()$, weights $w_{1:n}$, intrinsic dimension d, embedding dimension s

Initial coordinates $\mathbf{Y} \in \mathbb{R}^{n \times s}$, with $\mathbf{Y}_{k,:}$ representing the coordinates of point k.

Init : Compute Laplacian matrix $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{n \times n}$ using $\mathbf{X}$ and $K_h()$ by (2.4)

while *not converged* **do**

Compute $\mathbf{H} = [\mathbf{H}_k]_{k=1:n} \in \mathbb{R}^{n \times s \times s}$ the pushforward co-metric at data points from $\mathbf{Y}$ and $\boldsymbol{\mathcal{L}}$ using (2.7).

Compute $\text{Loss}(\mathbf{H}_{1:n})$ and $\nabla_{\mathbf{Y}} \text{Loss}(\mathbf{H})$ by (3.2.4) and (3.13)

Take a gradient step $\mathbf{Y} \leftarrow \mathbf{Y} - \eta \nabla_{\mathbf{Y}} \text{Loss}(\mathbf{H})$

end

Output: $\mathbf{Y}$

Algorithm 1: `Outline of the Riemannian Relaxation Algorithm`.

loss function in the second case.

3.2.1 The case $s = d$ (embedding dimension equals intrinsic dimension).

Under this condition, it can be shown [26] that $\phi : \mathcal{M} \to \mathbb{R}^d$ is an isometry iff g_p, $p \in \mathcal{M}$ system equals the unit matrix $\mathbf{I}_d$. For example, replacing g_p with the identify matrix in (2.1) returns the standard Euclidean dot product. Based on this observation, it is natural to measure the quality of the data embedding $\mathbf{Y}$ as the departure of the Riemannian metric obtained via (2.7) from the unit matrix.

This is the starting idea for the distortion measure we propose to optimize. We develop it further as follows. First, we choose to use the co-metric of g, evaluated by $\mathbf{H}$ instead of pushforward metric itself. Naturally $\mathbf{H}_k = \mathbf{I}_d$ iff $\mathbf{H}_k^{-1} = \mathbf{I}_d$, so the co-metric identifies isometry as well. When no isometric transformation exists, it is likely that optimizing with respect to g and optimizing with respect to h will arrive to different embeddings. There is no mathematically compelling reason, however, to prefer optimizing one over the other. We

choose to optimize with respect to h for several reasons:

1. It is computationally faster. Estimation of the pushforward Riemannian metric occurs through (2.7) which estimates the co-metric. To find the pushforward metric one must invert each of these matrices which can be computationally expensive.

2. It is numerically more stable. Depending on the condition number of the matrix $\mathbf{H}_k$ inversion can be numerically unstable. This is exacerbated in the case where $s > d$ since a psuedo-inverse must be taken instead.

3. In our experience users find $\mathbf{H}_k$ easier to interpret. This is since $\mathbf{H}_k$ represents the direction and amount of distortion as opposed to the scaling required to "correct" the the dot product (and consequently distances) between nearby vectors. When observed in a plot $\mathbf{H}_k$ shows how the space has been distorted rather than showing the correction that must be applied to remove the distortion. See figure 2.1 for demonstration.

Second, we choose to measure the distortion of $\mathbf{H}_k$ by $||\mathbf{H}_k - \mathbf{I}||$ where $||\,||$ denotes the matrix spectral norm. This choice will be motivated shortly.

Third we choose to assign each point a weight. We do not assume that the data are sampled uniformly from the manifold $\mathcal{M}$, instead we weight the points by an estimate of the sampling density. For this reason we choose the weights $w_{1:n}$ to be proportional to $\tilde{\mathbf{D}}$ from (2.4). As [14] show, these values converge to the sampling density π on $\mathcal{M}$.

Putting these together, we obtain the loss function

$$\text{Loss}(\mathbf{Y}; \boldsymbol{\mathcal{L}}, w) = \sum_{k=1}^{n} w_k \left|\left|\mathbf{H}_k - \mathbf{I}_d\right|\right|^2. \tag{3.1}$$

To motivate the choice of a "squared loss" instead of simply using $||\mathbf{H}_k - \mathbf{I}_d||$, notice that $||\,||$ is not differentiable at 0, but $||\,||^2$ is. A natural question to ask about Loss is if it is convex.

The following proposition summarizes a set of relevant convexity facts.

Proposition 13. *Denote by $\lambda_{1:d}(\boldsymbol{H}_k) \geq 0$ the eigenvalues of $\boldsymbol{H}_k$, in decreasing order and assume $\boldsymbol{Y}$ is in a compact, convex set. Then*

1. $\lambda_1(\boldsymbol{H}_k)$*,* $\lambda_1(\boldsymbol{H}_k) - \lambda_d(\boldsymbol{H}_k)$ *and* $\lambda_1(\boldsymbol{H}_k) - \sum_{d'=1}^{d} \lambda_{d'}(\boldsymbol{H}_k)$ *are convex in* $\boldsymbol{Y}$*.*

2. $||\boldsymbol{H}_k - \boldsymbol{I}_d||^2$ *is convex in* $\boldsymbol{Y}$ *whenever* $||\boldsymbol{H}_k - \boldsymbol{I}_d||$ *is convex and differentiable in* $\boldsymbol{Y}$*.*

Proof. The first follows since $\lambda_1(\mathbf{H}_k) = ||\mathbf{H}_k||_2^2$ and norms are convex, the rest follows by the sorting. To prove the second part we first prove the following Lemma.

Proposition 14. *If* $f : S \subseteq \mathbb{R}^D \to \mathbb{R}$ *is convex, non-negative and* $\nabla^2 f$ *exists for all* $x \in \operatorname{int} S$*, then* $\frac{1}{2} f^2(x)$ *is convex.*

Proof $\nabla\left(\frac{1}{2}f^2\right) = f\nabla f$; $\nabla^2\left(\frac{1}{2}f^2\right) = f\nabla^2 f + \nabla f \nabla f'$ which is positive definite whenever $f\nabla^2 f$ is. □.

Using the above Lemma, and the fact that $||\mathbf{H}_k - \mathbf{I}_n||$ is non-negative and infinitely differentiable almost everywhere we obtain the desired result. □

The Loss may not be convex near its minimum, but squaring the loss only improves convexity.

3.2.2 Choosing the right measure of distortion

Let us motivate the choice of norm and weights in the definition of the loss (3.1). The *norm of a Hermitian bilinear functional (i.e symmetric tensor of order 2)* $g : \mathbb{R}^s \times \mathbb{R}^s \to \mathbb{R}$ is defined as $\sup_{u \neq 0} |g(u,u)|/||u||$. In a fixed orthonormal base of $\mathbb{R}^s$, $g(u,v) = u'\mathbf{G}v$, $||g|| = \sup_{u\neq 0} |u'\mathbf{G}u|$. One can define norms with respect to any metric g_0 on $\mathbb{R}^s$ (where g_0 is represented in coordinates by $\mathbf{G}_0$, a symmetric, positive definite matrix), by $||u||_{\mathbf{G}_0} = u'\mathbf{G}_0 u$, respectively

$$\begin{aligned} ||g||_{\mathbf{G}_0} &= \sup_{u\neq 0} |u'\mathbf{G}u|/||u||_{\mathbf{G}_0} \\ &= \sup_{\tilde{u}\neq 0} |\tilde{u}'\mathbf{G}_0^{-1/2}\mathbf{G}\mathbf{G}_0^{-1/2}\tilde{u}|/||\tilde{u}|| \\ &= \lambda_{max}(\mathbf{G}_0^{-1/2}\mathbf{G}\mathbf{G}_0^{-1/2}) \end{aligned}$$

In particular, since any Riemannian metric at a point k is a g as above, by setting g and g_0 respectively to $\mathbf{H}_k$ and $\mathbf{I}_d$ we measure the *operator* norm of the distortion by $||\mathbf{H}_k - \mathbf{I}_d||$. In

other words, the appropriate operator norm we seek can be expresed as a matrix spectral norm.

The expected loss over the data set, given a distribution represented by the weights $w_{1:n}$ is then identical to the expression of Loss in (3.1). If the weights are computed as in (2.4), it is easy to see that the loss function in (3.1) is the finite sample version of the squared L_2 distance between h and g_0 on the space of Riemannian metrics on $\mathcal{M}$, with respect to base measure πdV_{g_0}

$$||h - g_0||^2_{g_0} \;=\; \int_{\mathcal{M}} ||h - g_0||^2_{g_0} \pi dV_{g_0}, \quad \text{with } dV_{g_0} \text{volume element on } \mathcal{M}. \tag{3.2}$$

with

$$||g||_{g_0}\big|_p \;=\; \sup_{u,v \in \mathcal{T}_p\mathcal{M}\setminus\{0\}} \frac{< x, y >_{g_p}}{< x, y >_{g_{0p}}}. \tag{3.3}$$

The right hand side of (3.3) above represents the *tensor norm* of g_p on $\mathcal{T}_p\mathcal{M}$ with respect to the Riemannian metric g_{0p}. Therefore, from the arguments above, Loss($\mathbf{Y}$) represents the discretized version of the expected distortion and is the natural discrete loss function for estimating deviation from isometry into Euclidean spaces when $s = d$.

3.2.3 Defining Loss *for embeddings with $s > d$ dimensions*

When $s > d$ the loss in (3.1) no longer encodes an isometric embedding. The embedding metric will not be the identity matrix in an isometric embedding into $\mathbb{R}^s$ since the resulting manifold $\mathcal{N} \subset \mathbb{R}^s$ must also be of dimension d, but the embedding co-metric $\mathbf{H}$ will be an $s \times s$ matrix with rank d as in (2.2). In order to extend the loss in the $s > d$ case to this case we have to define operator norms with respect to rank defective operators, as follows.

Consider $\mathbf{G}, \mathbf{G}_0 \in \mathbb{R}^{s\times s}$, two symmetric matrices with $\mathbf{G}_0$ positive semi-definite of rank $d < s$. We would like to extend the $\mathbf{G}_0$ norm of $\mathbf{G}$ to this case. We start with the family of norms $||||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}$ for $\epsilon > 0$ and we define

$$||\mathbf{G}||_{\mathbf{G}_0} \;=\; \lim_{\epsilon\to 0} ||\mathbf{G}||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}. \tag{3.4}$$

The following proposition highlights the behavior of $||||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}$ and $||||_{\mathbf{G}_0}$ and shows that the latter is also a matrix norm, that can take ∞ as a special value.

Proposition 15. *Let $\boldsymbol{G}, \boldsymbol{G}_0 \in \mathbb{R}^{s\times s}$ be symmetric matrices, with $\boldsymbol{G}_0$ positive semi-definite of rank $d < s$, and let $\epsilon > 0$, $\gamma(u,\varepsilon) = \frac{u'\boldsymbol{G}u}{u'\boldsymbol{G}_0u+\epsilon||u||^2}$. Then,*

1. $||\boldsymbol{G}||_{\boldsymbol{G}_0+\varepsilon\boldsymbol{I}_s} = ||\tilde{\boldsymbol{G}}||_2$ *with* $\tilde{\boldsymbol{G}} = (\boldsymbol{G}_0+\epsilon I)^{-1/2}\boldsymbol{G}(\boldsymbol{G}_0+\epsilon I)^{-1/2}$.

2. *If* $||\boldsymbol{G}||_{\boldsymbol{G}_0+\varepsilon\boldsymbol{I}_s} < r$, *then* $\lambda^\dagger(\boldsymbol{G}) < \epsilon r$ *with* $\lambda^\dagger(\boldsymbol{G}) = \sup_{v\in\mathrm{Null}(\boldsymbol{G}_0)}\gamma(v,\varepsilon)$,

3. $||||_{\boldsymbol{G}_0}$ *is a matrix norm that takes infinite values when* $\mathrm{Null}\,\boldsymbol{G}_0 \not\subseteq \mathrm{Null}\,\boldsymbol{G}$.

Proof.

$$
\begin{aligned}
||\mathbf{G}||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s} &= \sup_{u\neq 0}\frac{u'\mathbf{G}u}{u'\mathbf{G}_0u+\epsilon||u||^2} && (3.5)\\
&= \sup_{u\neq 0}\frac{v'\mathbf{G}_\epsilon^{-1}\mathbf{G}\mathbf{G}_\epsilon^{-1}v}{||v||^2} \quad \text{with } \mathbf{G}_\epsilon = (\mathbf{G}_0+\epsilon I)^{1/2} \text{ and } v = \mathbf{G}_\epsilon u && (3.6)\\
&= ||\tilde{\mathbf{G}}||_2 \quad \text{with } \tilde{\mathbf{G}} = (\mathbf{G}_0+\epsilon I)^{-1/2}\mathbf{G}(\mathbf{G}_0+\epsilon I)^{-1/2} && (3.7)
\end{aligned}
$$

For (2), we first prove the following fact

$$
\sup_{u\in\mathbb{R}^s}\frac{|u'\mathbf{G}u|}{u'\mathbf{G}_0u+\epsilon||u||^2}
\begin{cases}
= \sup_{u\in\mathrm{Null}\,\mathbf{G}^\perp}\frac{|u'\mathbf{G}u|}{u'\mathbf{G}_0u+\epsilon||u||^2} & \text{if } \mathrm{Null}(\mathbf{G}) = \mathrm{Null}(\mathbf{G}_0)\\
\leq \max_{\alpha^2+\beta^2=1}\frac{\beta^2\lambda^\dagger(\mathbf{G})+\alpha^2\lambda_{max}(\mathbf{G})+2\alpha\beta\Theta_{max}(\mathbf{G},\mathbf{G}_0)}{\beta^2\epsilon+\alpha^2(\lambda^*_{min}(\mathbf{G}_0)+\epsilon)} & \text{if } \mathrm{Null}(\mathbf{G}) \neq \mathrm{Null}(\mathbf{G}_0)
\end{cases}
\tag{3.8}
$$

where $\lambda_{max}(\mathbf{G})$ is the spectral radius of $\mathbf{G}$,

$$
\Theta(\mathbf{G},\mathbf{G}_0) = \sup_{||u||=||v||=1, v\in\mathrm{Null}\,\mathbf{G}, u\in\mathrm{Null}\,\mathbf{G}_0} u'\mathbf{G}v
$$

is the cosine of the principal angle between $\mathrm{Null}\,\mathbf{G}$ and $\mathrm{Null}\,\mathbf{G}_0$, and $\lambda^*_{min}(\mathbf{G}_0)$ is the smallest non-zero eigenvalue of $\mathbf{G}_0$.

Denote for simplicity

$$
g(u) = \frac{|u'\mathbf{G}u|}{u'\mathbf{G}_0u+\epsilon||u||^2}
$$

If $\text{Null}(\mathbf{G}) = \text{Null}(\mathbf{G}_0)$ then for $u \in \text{Null}\,\mathbf{G}$ the value is 0, which cannot be the sup. Let $u_1 = v \oplus u_0$ with $u_0 \in \text{Null}\,\mathbf{G}$, $v \in \text{Null}\,\mathbf{G}^{\perp}$. Then $u_1'\mathbf{G}_0 u_1 + \epsilon||u_1||^2 = v'\mathbf{G}_0 v + \epsilon||v||^2 + \epsilon||u_0||^2 > v'\mathbf{G}_0 v + \epsilon||v||^2$. Hence, the u which attains the supremum must be in $\text{Null}\,\mathbf{G}$.

Now note that, if $\text{Null}\,\mathbf{G} \neq \text{Null}\,\mathbf{G}_0$, $\mathbb{R}^s = \text{Null}\,\mathbf{G}_0 \oplus \text{Null}\,\mathbf{G}_0^{\perp}$, and $\text{Null}\,\mathbf{G}_0 = (\text{Null}\,\mathbf{G}_0 \cap \text{Null}\,\mathbf{G}) \oplus \mathcal{V}$, with $\mathcal{V}$ the orthogonal complement of $\text{Null}\,\mathbf{G}_0 \cap \text{Null}\,\mathbf{G}$ in $\text{Null}\,\mathbf{G}_0$ and the supremum of $g(u) =$ is attained on $\mathcal{U} = \mathcal{V} \oplus \text{Null}\,\mathbf{G}_0^{\perp}$ (as adding any component along the orthogonal complement of this space only adds a positive value to the denominator, increasing $g(u)$). Any $u \in \mathcal{U}$ can be written as $u = \alpha u_0 \oplus \beta v_0$ with $u_0 \in \text{Null}\,\mathbf{G}_0^{\perp}$ and $v_0 \in \mathcal{V}$ unit vectors. By upper bounding every term in the numerator and lower bounding $u_0'\mathbf{G}_0 u_0$ we obtain the result. Note that for ϵ small enough, the expression in 3.8 is close to $\frac{1}{\epsilon}\lambda^{\dagger}(\mathbf{G})$.

Then for (2), let $v \in \mathcal{V}$ and compute $g(v)$ as above, with $\alpha = 0$. It follows that $g(v) = \frac{|v'\mathbf{G}v|}{\epsilon||v||^2}$ and by taking the supremum over $v \in \mathcal{V}$ we obtain that $\sup_{\mathcal{V}} g(v) = \frac{1}{\epsilon}\lambda^{\dagger}(\mathbf{G}) < r$, from which the result follows.

For (3), it is obvious that when $\epsilon \to 0$, $g(v) \to \infty$ on $\mathcal{V}$, but remains finite for $u \notin \mathcal{V}$. More precisely, $||\mathbf{G}||_{\mathbf{G}_0} = \infty$ iff $\text{Null}\,\mathbf{G}_0 \not\subseteq \mathbf{G}$. To verify that $||||_{\mathbf{G}_0}$ is a norm, we must verify the triangle inequality, since the other two properties obviously hold. If $||\mathbf{A}||_{\mathbf{G}_0} = \infty$ or $||\mathbf{B}||_{\mathbf{G}_0} = \infty$, triangle inequality holds trivially. Assume then that $||\mathbf{A}||_{\mathbf{G}_0}, ||\mathbf{B}||_{\mathbf{G}_0} < \infty$. Since $||\mathbf{A}||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s} + ||\mathbf{B}||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s} \geq ||\mathbf{A}+\mathbf{B}||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}$ for every $\epsilon > 0$, then in the limit we will have that $||\mathbf{A}||_{\mathbf{G}_0} + ||\mathbf{B}||_{\mathbf{G}_0} \geq ||\mathbf{A}+\mathbf{B}||_{\mathbf{G}_0}$. □

In other words, the appropriate operator norm we seek can be expressed as a (generalized) matrix spectral norm. In our cases $\mathbf{G}_0 = \mathbf{I_d}$ and $\mathbf{G} = \mathbf{H}_k - \mathbf{I}_d$. Hence, $||\,||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}$ can be computed as the spectral norm of a matrix. The computation of $||\,||_{\mathbf{G}_0}$ is similar, with the additional step of checking first if $\text{Null}\,\mathbf{G}_0 \not\subseteq \text{Null}\,\mathbf{G}$, in which case we output the value ∞. Let $B_\epsilon(\mathbf{0}, r)$ ($B(\mathbf{0}, r)$) denote the r-radius ball centered at $\mathbf{0}$ in the $||\,||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}$ ($||\,||_{\mathbf{G}_0}$). From Proposition 15 it follows that if $\mathbf{G} \in B_\epsilon(\mathbf{0}, r)$ then $\lambda^{\dagger}(\mathbf{G}) < \epsilon r$ and if $\mathbf{G} \in B(\mathbf{0}, r)$ then $\text{Null}(\mathbf{G}_0) \subseteq \text{Null}(\mathbf{G})$. In particular, if $\text{rank}\,\mathbf{G} = \text{rank}\,\mathbf{G}_0$ then $\text{Null}(\mathbf{G}) = \text{Null}(\mathbf{G}_0)$.

Putting this all together, to define the loss for $s > d$ we set $\mathbf{G} = \mathbf{H}_k$ and $\mathbf{G}_0 = \mathbf{U}_k\mathbf{U}_k'$, with

$\mathbf{U}_k$ an orthonormal basis for $\mathcal{T}_k\mathcal{M}$ the tangent subspace at k. The norms $||\,||_{\mathbf{G}_0+\varepsilon\mathbf{I}_s}, ||\,||_{\mathbf{G}_0}$ act as soft and hard barrier functions constraining the span of $\mathbf{H}_k$ to align with the tangent subspace of the data manifold.

$$\text{Loss}(\mathbf{Y};\, \boldsymbol{\mathcal{L}},\, w,\, d,\, \varepsilon_{orth}) \;=\; \sum_{k=1}^{n} w_k || \underbrace{(\mathbf{U}_k\mathbf{U}_k' + \varepsilon_{orth}^2\mathbf{I}_s)^{-1/2}\,(\mathbf{H}_k - \mathbf{U}_k\mathbf{U}_k')\,(\mathbf{U}_k\mathbf{U}_k' + \varepsilon_{orth}^2\mathbf{I}_s)^{-1/2}}_{\tilde{\mathbf{G}}_k} ||^2. \tag{3.9}$$

We note that for $s = d$ and $\varepsilon_{orth} = 0$ (3.2.4) and (3.1) are equivalent, so that (3.2.4) is a generalization of (3.1). In practice we estimate $\mathbf{U}_k$ as the principal d eigen-vectors of $\mathbf{H}_k$ so that $\mathbf{U}_k\mathbf{U}_k'$ is a rank d matrix with a unit-length basis and it is this matrix to which we are comparing $\mathbf{H}_k$. We are therefore encouraging $\mathbf{H}_k$ to have rank d as it will be a d-dimensional sub-manifold of $\mathbb{R}^s$ and that it have unit scaling in any direction (i.e. no stretch/shrinking). Finally we ensure that the norm is a proper matrix norm but with respect to a rank-deficient base matrix – the target.

3.2.4 Properties of the Loss Function

Before we go on we prove some desirable properties of the loss function. In particular that it is invariant to location shifts and rotations. Since these transformations are themselves isometries these are desirable properties when measuring isometry.

Crucial to this work, however, is the following equivalent formula for the Riemannian Metric as estimated in 2.7. Let $\boldsymbol{\mathcal{L}}_k$ denote the kth row of $\boldsymbol{\mathcal{L}}$, then $\mathbf{H}_k$ can be rewritten in the convenient form

$$\mathbf{H}_k(\mathbf{Y}) \;=\; \frac{1}{2}\mathbf{Y}'[\text{diag}(\boldsymbol{\mathcal{L}}_{k:}) - (e_k e_k'\boldsymbol{\mathcal{L}}) - (e_k e_k'\boldsymbol{\mathcal{L}})']\mathbf{Y} \;\equiv\; \frac{1}{2}\mathbf{Y}'\mathbf{L}_k\mathbf{Y} \tag{3.10}$$

where e_k refers to the kth standard basis vector of $\mathbb{R}^n$ and $\mathbf{L}_k$ is a symmetric matrix precomputed from entries in $\boldsymbol{\mathcal{L}}$, $\mathbf{L}_k = [\text{diag}(\boldsymbol{\mathcal{L}}_{k:}) - (e_k e_k'\boldsymbol{\mathcal{L}}) - (e_k e_k'\boldsymbol{\mathcal{L}})']$; $\mathbf{L}_k$ has non-zero rows only for the neighbors of k. We also prove the additional fact that $\mathbf{L}_k$ is positive semi-definite which will be useful later:

Proposition 16. *$\boldsymbol{L}_k$ is positive semi-definite.*

Proof. Let $\mathbf{x} = [x_1, \ldots, x_n]$ so that

$$\begin{aligned}
\mathbf{x}'\mathbf{L}_k\mathbf{x} &= x_1\left(\boldsymbol{\mathcal{L}}_k^1 x_1 - \boldsymbol{\mathcal{L}}_k^1 x_k\right) + x_2\left(\boldsymbol{\mathcal{L}}_k^2 x_2 - \boldsymbol{\mathcal{L}}_k^2 x_k\right) + \cdots + x_{k-1}\left(\boldsymbol{\mathcal{L}}_k^{k-1} x_{k=1} - \boldsymbol{\mathcal{L}}_k^{k-1} x_k\right) \\
&\quad + x_k\left(-\sum_{i=1}^{n} \boldsymbol{\mathcal{L}}_k^i x_i\right) + x_{k+1}\left(\boldsymbol{\mathcal{L}}_k^{k+1} x_{k+1} - \boldsymbol{\mathcal{L}}_k^{k+1} x_k\right) + \cdots + x_n\left(\boldsymbol{\mathcal{L}}_k^n x_n - \boldsymbol{\mathcal{L}}_k^n x_k\right) \\
&= \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i x_i^2 - x_k \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i x_i - x_k \sum_{i=1}^{n} \boldsymbol{\mathcal{L}}_k^i x_i \\
&= \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i [x_i^2 - 2x_k x_i] - \boldsymbol{\mathcal{L}}_k^k x_k^2 \\
&= \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i [(x_i - x_k)^2 - x_k^2] - \boldsymbol{\mathcal{L}}_k^k x_k^2 \\
&= \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i (x_i - x_k)^2 - x_k^2 \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i - \boldsymbol{\mathcal{L}}_k^k x_k^2 \\
&= \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i (x_i - x_k)^2 - x_k^2(-\boldsymbol{\mathcal{L}}_k^k) - \boldsymbol{\mathcal{L}}_k^k x_k^2 \\
&= \sum_{i\neq k} \boldsymbol{\mathcal{L}}_k^i (x_i - x_k)^2 \\
&\geq 0
\end{aligned}$$

since $\mathbf{x}$ was arbitrary $\mathbf{L}_k$ is positive semidefinite (here equality holds for $x = \mathbf{1}$). □

The Loss is Invariant to Location Shifts

By equation (3.2.4) we have that the loss function interacts with $\mathbf{Y}$ through the estimated embedding co-metric $\mathbf{H}_k$ and by we have that $\mathbf{H}_k$ can be computed from $\mathbf{Y}$ via the following function:

$$\mathbf{H}_k = H_k(\mathbf{Y}) = \mathbf{Y}'\mathbf{L}_k\mathbf{Y}$$

Such that if $H_k(\mathbf{Y}) = H_k(\mathbf{Y} + \mathbf{C})$ for any constant location shift matrix $\mathbf{C} = [c_1\mathbf{1} \ldots c_d\mathbf{1}]$. Then the loss function from will also be the same: $\text{Loss}(\mathbf{Y}) = L(\mathbf{Y} + \mathbf{C})$ and the loss function is invariant to location shifts.

Proposition 17. $H_k(\boldsymbol{Y}) = H_k(\boldsymbol{Y} + \boldsymbol{C})$

Proof. For simplicity we drop the k index.

$$\begin{aligned} H(\mathbf{Y}+\mathbf{C}) &= (\mathbf{Y}+\mathbf{C})'\mathbf{L}(\mathbf{Y}+\mathbf{C}) \\ &= \mathbf{Y}'\mathbf{L}\mathbf{Y} + \mathbf{C}'\mathbf{L}\mathbf{Y} + \mathbf{Y}'\mathbf{L}\mathbf{C} + \mathbf{C}'\mathbf{L}\mathbf{C} \\ &= \mathbf{Y}'\mathbf{L}\mathbf{Y} + (\mathbf{L}\mathbf{C})'\mathbf{Y} + \mathbf{Y}'(\mathbf{L}\mathbf{C}) + \mathbf{C}'(\mathbf{L}\mathbf{C}) \qquad \text{by symmetry of } \mathbf{L} \end{aligned}$$

Let's take a closer look at $\mathbf{LC}$. Recall that $\mathbf{1}$ is in the *null space* of $\mathbf{L}$:

$$\mathbf{L}\mathbf{C} = \mathbf{L}[c_1\mathbf{1}, \dots c_d\mathbf{1}] = [c_1\mathbf{L}\mathbf{1}, \dots, c_d\mathbf{L}\mathbf{1}] = [c_1\mathbf{0}, \dots, c_d\mathbf{0}] = \mathbf{0}$$

So that

$$\begin{aligned} H(\mathbf{Y}+\mathbf{C}) &= \mathbf{Y}'\mathbf{L}\mathbf{Y} + (\mathbf{L}\mathbf{C})'\mathbf{Y} + \mathbf{Y}'\mathbf{L}\mathbf{C} + \mathbf{C}'\mathbf{L}\mathbf{C} \\ &= \mathbf{Y}'\mathbf{L}\mathbf{Y} + (\mathbf{0})'\mathbf{Y} + \mathbf{Y}'\mathbf{0} + \mathbf{C}'\mathbf{0} \\ &= \mathbf{Y}'\mathbf{L}\mathbf{Y} = H(\mathbf{Y}) \end{aligned}$$

As claimed. □

The Loss is Invariant to Orthogonal Transformations

Let $\mathbf{B}$ be some orthonormal (rotation) matrix in $\mathbb{R}^{s\times s}$ where s is the dimension of $\mathbf{Y}$. $\mathbf{B}$ is such that $\mathbf{B}\mathbf{B}' = \mathbf{B}'\mathbf{B} = \mathbf{I}$. If $\ell(\mathbf{Y}\mathbf{B}) = \ell(\mathbf{Y})$ where ℓ denotes a single term of the loss function, then we have $\text{Loss}(\mathbf{Y}\mathbf{B}) = \text{Loss}(\mathbf{Y})$.

Proof. Let $\{\lambda_1, \dots, \lambda_d\}$ be the eigenvalues of $\mathbf{Y}'\mathbf{L}\mathbf{Y}$ and let the eigenvalues of $(\mathbf{Y}\mathbf{B})'\mathbf{L}(\mathbf{Y}\mathbf{B})$ be given by $\{\tilde{\lambda}_1, \dots, \tilde{\lambda}_d\}$ We will prove the claim by showing that $\{\lambda_1, \dots, \lambda_d\} = \{\tilde{\lambda}_1, \dots, \tilde{\lambda}_d\}$. First denote $\mathbf{H} = \mathbf{Y}'\mathbf{L}\mathbf{Y}$ we have that $\mathbf{H}$ is symmetric and positive semi-definite by construction. Then note that: $(\mathbf{Y}\mathbf{B})'\mathbf{L}(\mathbf{Y}\mathbf{B}) = \mathbf{B}'\mathbf{Y}'\mathbf{L}\mathbf{Y}\mathbf{B} = \mathbf{B}'\mathbf{H}\mathbf{B}$. Since $\mathbf{H}$ is symmetric positive semi-definite we can write its eigendecomposition as: $\mathbf{H} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}'$ where $\mathbf{Q}$ is orthonormal and $\mathbf{\Lambda}$ is diagonal where the diagonal entries are the eigenvalues of $\mathbf{H}$. Then we have:

$$(\mathbf{Y}\mathbf{B})'\mathbf{L}(\mathbf{Y}\mathbf{B}) = \mathbf{B}'\mathbf{H}\mathbf{B} = \mathbf{B}'\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}'\mathbf{B} = (\mathbf{B}'\mathbf{Q})\mathbf{\Lambda}(\mathbf{Q}'\mathbf{B}) \equiv \tilde{\mathbf{Q}}\mathbf{\Lambda}\tilde{\mathbf{Q}}'$$

Then since $\tilde{\mathbf{Q}}'\tilde{\mathbf{Q}} = (\mathbf{B}'\mathbf{Q})'(\mathbf{B}'\mathbf{Q}) = \mathbf{Q}'\mathbf{B}\mathbf{B}'\mathbf{Q} = \mathbf{Q}'\mathbf{I}'\mathbf{Q} = \mathbf{Q}'\mathbf{Q} = \mathbf{I}$ and $\tilde{\mathbf{Q}}\tilde{\mathbf{Q}}' = (\mathbf{B}'\mathbf{Q})(\mathbf{B}'\mathbf{Q})' = \mathbf{B}'\mathbf{Q}\mathbf{Q}'\mathbf{B} = \mathbf{B}'\mathbf{I}'\mathbf{B} = \mathbf{B}'\mathbf{B} = \mathbf{I}$ We have that $\tilde{\mathbf{Q}}$ is orthonormal and therefore: $\mathbf{B}'\mathbf{H}\mathbf{B} = \tilde{\mathbf{Q}}\mathbf{\Lambda}\tilde{\mathbf{Q}}'$ Is the eigendecomposition of $\mathbf{B}'\mathbf{H}\mathbf{B}$ And so $\mathbf{\Lambda}$ is the diagonal matrix containing the eigenvalues of $\mathbf{B}'\mathbf{H}\mathbf{B}$ but this is the same diagonal matrix as the eigendecomposition of $\mathbf{H}$ and so they have the same eigenvalues, that is, $\{\lambda_1, \ldots, \lambda_d\} = \{\tilde{\lambda}_1, \ldots, \tilde{\lambda_d}\}$ as required. □

3.3 Riemannian Relaxation Algorithm

We are now ready to flesh out the algorithm outlined in 3.1. We optimize the loss (3.1) or (3.2.4) by projected gradient descent with line search. The projection consists of imposing $\sum_k \mathbf{Y}_k = 0$, which we enforce by centering $\nabla\mathbf{Y}$ before taking a step. This eliminates the degeneracy of the Loss in (3.1) and (3.2.4) with respect to constant shift in $\mathbf{Y}$.

To further improve the good trade-off between time per iteration and number of iterations, we found that a heavy-ball method with parameter α is effective. The heavy ball update rule is given by:

$$\mathbf{Y}_{t+1} = \mathbf{Y}_t - \eta_t \sum_{i=0}^{t} \alpha^i \nabla \operatorname{Loss}(\mathbf{Y}_{t-i}) \tag{3.11}$$

When embedding in $s > d$ dimensions, the loss function depends at each point k on finding the d-dimensional subspace $\mathbf{U}_k$. Mathematically, this subspace coincides with the span of the Jacobian $D\mathbf{Y}_k$ which can be identified with the d-principal subspace of $\mathbf{H}_k$. When computing the gradient of Loss we assume that $\mathbf{U}_{1:n}$ are fixed. Since the derivatives with respect to $\mathbf{Y}$ are taken only of $\mathbf{H}$ and not of the tangent subspace $\mathbf{U}_k$, algorithm 2 `RiemannianRelaxation` is actually an alternate minimization algorithm, which reduces the cost with respect to $\mathbf{Y}$ in one step, and with respect to $\mathbf{U}_{1:n}$ in the alternate step.

The Laplacian matrix $\boldsymbol{\mathcal{L}}$ is the renormalized graph Laplacian of [14] which has been proved to be an consistent estimator of the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ on $\mathcal{M}$ as discussed in chapter 2.

It was shown that although this $\boldsymbol{\mathcal{L}}$ is not a symmetric matrix, it converges to a normal operator in the limit $\epsilon \to 0$. The reader can easily verify, that in our implementation, only the

Laplacian rows are used, never the columns, hence the row normalization in (2.4) amounts to a rescaling at each point k. It is also known [14] that the renormalization weights $\tilde{\mathbf{D}}$ represent unbiased estimators of the sampling density on $\mathcal{M}$. These are the values we input to Algorithm 2 for the weights w. Next in order to minimize the loss functions via gradient descent we must compute the derivative of the loss functions.

3.3.1 Computing ∇ Loss

We note that the loss is a scalar function of a matrix and so in particular our gradient will itself be a matrix of the same size as the embedding ($n \times s$) whose entries are the partial derivatives of the real-valued loss function with respect to that element of the matrix. To compute the gradient of a real-valued function of a matrix we follow [31].

Proposition 18. *Let* Loss_k *denote term* k *of* Loss. *If* $s = d$, *the gradient of* Loss_k *as given by* (3.1) *is*

$$\frac{\partial \text{Loss}_k}{\partial \boldsymbol{Y}} = 2w_k\lambda_k^*\boldsymbol{L}_k\boldsymbol{Y}\mathbf{u}_k\mathbf{u}_k', \tag{3.12}$$

with λ_k^* *the largest eigenvalue of* $\boldsymbol{H}_k - \boldsymbol{I}_d$ *and* $\mathbf{u}_k$ *is the corresponding eigenvector.*
If $s > d$, *the gradient of* Loss_k *of* (3.2.4) *is*

$$\frac{\partial \text{Loss}_k}{\partial \boldsymbol{Y}} = 2w_k\lambda_k^*\boldsymbol{L}_k\boldsymbol{Y}\boldsymbol{\Pi}_k\mathbf{u}_k\mathbf{u}_k'\boldsymbol{\Pi}_k' \tag{3.13}$$

where $\boldsymbol{\Pi}_k = (\boldsymbol{U}_k\boldsymbol{U}_k' + (\varepsilon_{orth})_k\boldsymbol{I}_s)^{-1/2}$, λ_k^* *is the largest eigenvalue of* $\tilde{\boldsymbol{G}}_k$ *of* (3.2.4) *and* $\mathbf{u}_k$ *is the corresponding eigenvector.*

At each iteration computing the gradient is $\mathcal{O}((S + s^3)n)$ where S is the number of nonzero entries of $\boldsymbol{\mathcal{L}}$.

Proof. First, as stated in the previous section we will consider the $\mathbf{U}_k$ fixed and update them after updating $\mathbf{Y}$. With this in mind we can simplify the loss by collecting terms we consider fixed into a matrix we call Π and then expand the loss function. That is, we can write the loss in (3.2.4) as:

$$\sum_{k=1}^{n}\left|\left|\frac{1}{2}\boldsymbol{\Pi}_k'\mathbf{Y}'\mathbf{L}_k\mathbf{Y}\boldsymbol{\Pi}_k - \boldsymbol{\Pi}_k\mathbf{U}_k\mathbf{U}_k\boldsymbol{\Pi}_k\right|\right|_2^2$$

Where $\mathbf{\Pi}_k = (\mathbf{U}_k\mathbf{U}_k' + (\varepsilon_{orth})_k\mathbf{I}_s)^{-1/2}$. We take the $\mathbf{\Pi}_k$ matrices to be fixed and don't depend on the data points $\mathbf{Y}$ (in practice they do, however, after taking a gradient step we update the $\mathbf{\Pi}_k$ in an alternating minimization style algorithm).

Since the loss function when $s > d$ i.e. (3.2.4) reduces to the loss of (3.1) by taking since we can take $\mathbf{U}_k\mathbf{U}_k' = \mathbf{\Pi}_k = I_d$ and $\varepsilon_{orth} = 0$. Therefore compute the derivative of (3.2.4), when $s > d$, without loss of generality.

Next, since the derivative is a linear operator it is sufficient to show that the derivative of a single term of the sum in the loss function is of the form:

$$\frac{\partial \operatorname{Loss}_k}{\partial Y} = (2|\lambda_k^*|)\operatorname{sgn}(\lambda_k^*)\mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k\mathbf{u}_k'\mathbf{\Pi}_k'$$

In order to write the derivative of the loss function we will apply the chain rule. In particular we have several functions which will be matrix-valued and some which are scalar valued, again we follow [31] which tells us how to combine these in an appropriate chain rule, at the end we will return to a gradient. To use the chain rule we first define the function l_k as a composition of functions:

$$\operatorname{Loss}_k(\mathbf{Y}) \equiv \rho(P_k(H_k(\mathbf{Y})) - \mathbf{C}_k) \tag{3.14}$$

With $\mathbf{C}_k = \mathbf{\Pi}_k\mathbf{U}_k\mathbf{U}_k\mathbf{\Pi}_k$ and

$$\rho(\mathbf{U}) = (\max_k |\lambda_k(\mathbf{U})|)^2$$
$$P_k(\mathbf{H}) = \mathbf{\Pi}_k'\mathbf{H}\mathbf{\Pi}_k$$
$$H_k(\mathbf{Y}) = \frac{1}{2}\mathbf{Y}'\mathbf{L}_k\mathbf{Y}$$

Where $\mathbf{U}, \mathbf{H}$ are both symmetric. Here we note that the matrix spectral norm reduces to the spectral radius if $\mathbf{U}$ is symmetric. Since $H_k(\mathbf{Y})$ is defined to be symmetric and $\mathbf{C}_k$ is symmetric this is the case. By the chain rule:

$$D\operatorname{Loss}_k(\mathbf{Y}) = D\rho(P_k(H_k(\mathbf{Y})) - \mathbf{C}_k)DP_k(H_k(\mathbf{Y}))DH_k(\mathbf{Y}) \tag{3.15}$$

Taking these from left to right: Since ρ is defined to be the largest (in absolute value) eigenvalue of $\mathbf{U}$ (squared) the derivative is the kronecker product between the corresponding

eigenvector and itself multiplied by the sign of the eigenvalue (see [31]):

$$D\sqrt{\rho(\mathbf{U})} = sgn(\lambda_k^*)(\mathbf{u}_k' \otimes \mathbf{u}_k')$$

Where $|\lambda_k^*| = \sqrt{\rho(\mathbf{U})}$ and $\mathbf{U}\mathbf{u}_k = \lambda_k^*\mathbf{u}_k$ Then since we square the spectral radius we add the factor of $(2|\lambda_k^*|)$ so that:

$$D(\rho(\mathbf{U}) = (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{u}_k' \otimes \mathbf{u}_k')$$

Next we have that:

$$DP_k(\mathbf{H}) = (\mathbf{\Pi}_k' \otimes \mathbf{\Pi}_k')$$

Since:

$$\begin{aligned} P_k(\mathbf{H}) &= \mathbf{\Pi}_k'\mathbf{H}\mathbf{\Pi}_k \\ dP_k(\mathbf{H}) &= \mathbf{\Pi}_k' d\mathbf{H}\mathbf{\Pi}_k \\ \Rightarrow vec(dP_k(\mathbf{H})) &= vec(\mathbf{\Pi}_k' d\mathbf{H}\mathbf{\Pi}_k) \\ &= (\mathbf{\Pi}_k' \otimes \mathbf{\Pi}_k')\mathrm{d}vec(\mathbf{H}) \end{aligned}$$

Then we take the derivative of the co-metric with respect to the data:

$$DH_k(\mathbf{Y}) = \mathbf{N}_s(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)$$

Where $\mathbf{N}_s = \mathbf{I}_{s^2} + \mathbf{K}_{ss}$ for $\mathbf{K}_{ss}$ the commutation matrix defined in [31]. Since:

$$\begin{aligned} H_k(\mathbf{Y}) &= \frac{1}{2}\mathbf{Y}'\mathbf{L}_k\mathbf{Y} \\ \Rightarrow dH_k(\mathbf{Y}) &= \frac{1}{2}[(d\mathbf{Y})'\mathbf{L}_k\mathbf{Y} + \mathbf{Y}'\mathbf{L}_k d\mathbf{Y}] \\ \Rightarrow vec(dH_k(\mathbf{Y})) &= \frac{1}{2}[(\mathbf{Y}'\mathbf{L}_k' \otimes \mathbf{I}_s)dvec(\mathbf{Y}') + (\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)dvec(\mathbf{Y})] \\ &= \frac{1}{2}[(\mathbf{Y}'\mathbf{L}_k' \otimes \mathbf{I}_s)\mathbf{K}_{ns}dvec(\mathbf{Y}) + (\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)dvec(\mathbf{Y})] \\ &= \frac{1}{2}[\mathbf{K}_{ss}(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k')dvec(\mathbf{Y}) + (\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)dvec(\mathbf{Y})] \\ &= \frac{1}{2}[(\mathbf{K}_{ss} + \mathbf{I}_{s^2})(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)dvec(\mathbf{Y})] \qquad \mathbf{L}_k \text{ is symmetric} \\ &= \frac{1}{2}[2\mathbf{N}_s(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)dvec(\mathbf{Y})] \\ &= \mathbf{N}_s(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)dvec(\mathbf{Y}) \end{aligned}$$

Putting it all together using the chain-rule in (3.15):

$$Dl_k(\mathbf{Y}) = (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{u}_k' \otimes \mathbf{u}_k')(\mathbf{\Pi}_k' \otimes \mathbf{\Pi}_k')\mathbf{N}_s(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k) = vec\left(\frac{\partial l_k}{\partial Y}\right)'$$

We note that this is in the format of as the gradient of the loss as a function of $vec(\mathbf{Y})$ whereas since the function is real-valued we want to get at the gradient of $\mathbf{Y}$ such that the entries are the partial derivatives with respect to that portion of the matrix. We can simplify the above to get the claim:

$$\frac{\partial l_k}{\partial Y} = (2|\lambda_k^*|)\text{sgn}(\lambda_k^*)\mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k\mathbf{u}_k'\mathbf{\Pi}_k'$$

as follows:

Proof.

$$\begin{aligned}
D\,\text{Loss}_k(\mathbf{Y}) &= (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{u}_k' \otimes \mathbf{u}_k')(\mathbf{\Pi}_k' \otimes \mathbf{\Pi}_k')\mathbf{N}_s(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k) \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{u}_k' \otimes \mathbf{u}_k')(\mathbf{\Pi}_k' \otimes \mathbf{\Pi}_k')\frac{1}{2}(\mathbf{K}_{ss} + \mathbf{I}_{s^2})(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k) \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)\frac{1}{2}(\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k')(\mathbf{K}_{ss} + \mathbf{I}_{s^2})(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k) \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)\frac{1}{2}\left[(\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k')\mathbf{K}_{ss}(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k) + (\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k')(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)\right] \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)\frac{1}{2}\left[(\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k')(\mathbf{Y}'\mathbf{L}_k \otimes \mathbf{I}_s)\mathbf{K}_{ns} + (\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k')(\mathbf{I}_s \otimes \mathbf{Y}'\mathbf{L}_k)\right] \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)\frac{1}{2}\left[(\mathbf{u}_k'\mathbf{\Pi}_k'\mathbf{Y}'\mathbf{L}_k \otimes \mathbf{u}_k'\mathbf{\Pi}_k')\mathbf{K}_{ns} + (\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k'\mathbf{Y}'\mathbf{L}_k)\right] \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)\frac{1}{2}\left[\mathbf{K}_{11}(\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k'\mathbf{Y}'\mathbf{L}_k) + (\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k'\mathbf{Y}'\mathbf{L}_k)\right] \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{u}_k'\mathbf{\Pi}_k' \otimes \mathbf{u}_k'\mathbf{\Pi}_k'\mathbf{Y}'\mathbf{L}_k) && \mathbf{K}_{11} = 1 \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{\Pi}_k\mathbf{u}_k \otimes \mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k)'
\end{aligned}$$

Then note that:

$$\begin{aligned}
vec((2|\lambda_k^*|)sgn(\lambda_k^*)\mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k\mathbf{u}_k'\mathbf{\Pi}_k') &= (2|\lambda_k^*|)sgn(\lambda_k^*)vec([\mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k][1][\mathbf{u}_k'\mathbf{\Pi}_k']) \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{\Pi}_k\mathbf{u}_k \otimes \mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k)vec(1) \\
&= (2|\lambda_k^*|)sgn(\lambda_k^*)(\mathbf{\Pi}_k\mathbf{u}_k \otimes \mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k) \\
&= (Dl_k(\mathbf{Y}))'
\end{aligned}$$

So that

$$\frac{\partial l_k}{\partial Y} = (2|\lambda_k^*|)\mathrm{sgn}(\lambda_k^*)\mathbf{L}_k\mathbf{Y}\mathbf{\Pi}_k\mathbf{u}_k\mathbf{u}_k'\mathbf{\Pi}_k'$$

The proposition then follows by removing the absolute value and multiplication by the sign. This gives us the forumal in (3.13). □

Finally, by taking $\mathbf{U}_k\mathbf{U}_k' = \mathbf{\Pi}_k = I_d$ and $\varepsilon_{orth} = 0$ we arrive at the derivative in (3.12). □

Input : data $\mathbf{X}$, kernel function $K_h()$, initial coordinates $\mathbf{Y}^0$, weights $w_{1:n}$, intrinsic dimension d, orthonormal tolerance ε_{orth}, heavy ball parameter $\alpha \in [0, 1)$

Init : Compute: graph Laplacian $\boldsymbol{\mathcal{L}}$ by (2.4), matrices $\mathbf{L}_{1:n}$ as in (2.7). Set $\mathbf{S} = 0$

while *not converged* **do**

Compute ∇ Loss:

for *all* k **do**

1. Calculate $\mathbf{H}_k$ via (2.7);
2. **If** $s > d$
   (a) Compute $\mathbf{U}_k$ as the d principal eigenvectors of $\mathbf{H}_k$;
   (b) Compute gradient of $\nabla \mathrm{Loss}_k(\mathbf{Y})$ using (3.13);
3. **Else** ($s = d$): calculate gradient $\nabla \mathrm{Loss}_k(\mathbf{Y})$ using (3.12);
4. Add $\nabla \mathrm{Loss}_k(\mathbf{Y})$ to the total gradient;

end

Take a step in $\mathbf{Y}$:

1. Compute projected direction $\mathbf{S} \leftarrow (\mathbf{I}_n - e_n e_n')\nabla \mathrm{Loss} + \alpha\mathbf{S}$;
2. Find step size η by line search and update $\mathbf{Y} \leftarrow \mathbf{Y} - \eta\mathbf{S}$;

end

Output: $\mathbf{Y}$

Algorithm 2: `RiemannianRelaxation` (RR)

3.3.2 Riemannian Relaxation For large or noisy data

Here we describe an extension of the RR Algorithm which can naturally adapt to large or noisy data, where the manifold assumption holds only approximately. The idea is to sub-sample the data, but in a highly non-uniform way that improves the estimation of the geometry.

A simple preliminary observation is that, when an embedding is smooth, optimizing the loss on a subset of the data will be sufficient. Let $\mathcal{I} \subset \{1, \dots n\}$ be a set of size $n' < n$. The sub-sampled loss $\text{Loss}_{\mathcal{I}}$ will be computed only for the points $k' \in \mathcal{I}$. Note that the gradient of the term $||\mathbf{H}_k - \mathbf{I}_d||^2$ with respect to $\mathbf{Y}$ is non-zero on all neighbors of point k, and consequently they will be updated, even if there is no term in Loss for their Riemannian metric. If every point k has a sufficient number of neighbors in $\mathcal{I}$, this assures that the gradient of $\text{Loss}_{\mathcal{I}}$ will be a good approximation of ∇Loss at point k, even if $k \notin \mathcal{I}$, and does not have a term containing $\mathbf{H}_k$ in $\text{Loss}_{\mathcal{I}}$. If the sample $\mathcal{I}$ is uniformly randomly chosen at each iteration, we will be effectively implementing a stochastic gradient descent algorithm for this objective function. There would be a bias in this algorithm because of the separate estimation of $\mathbf{U}_k$. This highlights a more generic and well known advantage of iterative methods. They are more naturally amenable to simple and effective approximations than one-shot black-box methods like eigendecomposition or Semidefinite Programming. To optimize $\text{Loss}_{\mathcal{I}}$ by RR, it is sufficient to run the "for" loop in 2 over $k' \in \mathcal{I}$.

Algorithm PCS-RR describes how we choose a "good" subsample $\mathcal{I}$, with the help of the PrincipalCurves algorithm of [39]. Informally speaking, PrincipalCurves uses a form of Mean-Shift to obtain points in the d-dimensional manifold of highest density in the data. The result is generally biased, however [20] have shown that this algorithm offers a very advantageous bias-variance trade-off in case of manifolds with noise. We use the output $\hat{\mathbf{Y}}$ of PrincipalCurves to find a subset of points that (1) lie in a high density region relative to most directions in $\mathbb{R}^D$ and (2) are "in the middle" of their neighbors, or more formally, have neighborhoods of dimension at least d. In other words, this is a good heuristic to avoid

"border effects", or other regions where the d-manifold assumption is violated.

Input : data $\mathbf{X}$, kernel function $K_h()$, initial coordinates $\mathbf{Y}^0$, intrinsic dimension d, subsample size n', other parameters for RR

Compute $\hat{\mathbf{X}} = \text{PRINCIPALCURVES}(\mathbf{X}, K_h, d)$

Take a uniform sample $\mathcal{I}_0$ of size n' from $\{1, \dots n\}$ (without replacement).

for k' *in* $\mathcal{I}_0$ **do**

  Find $\mathbf{X}_l$ the nearest neighbor in $\mathbf{X}$ of $\hat{\mathbf{X}}_{k'}$, and add l to $\mathcal{I}$ (removing duplicates)

end

Output: $\mathbf{Y} = \text{RR}(\mathbf{Y}^0, K_h, d, \mathcal{I}, \dots)$

Algorithm 3: `PrincipalCurves-RiemannianRelaxation` (PCS-RR)

3.4 Experimental Results

In this section we evaluate the Riemannian Relaxation algorithm on both synthetic and real data. In the synthetic case we can control otherwise unknown parameters. Specifically we can control the intrinsic dimension d and the embedding dimension s for which an isometry exists and can control if $s = d$ or $s > d$. We also have the true isometric embedding which gives us a ground truth to which we can compare algorithms in otherwise ideal circumstances. On the other hand, the point of manifold learning is to be applied to real data sets and in particular, as we have discussed in previous chapters, to apply them to large data sets. We use real data as well to motivate the algorithm but also to display the ability for PCS-RR to scale to large data sets.

3.4.1 Results on Synthetic Data

In this section we consider several scenarios. Since Riemannian Relaxation accepts both an embedding dimension (s) and an intrinsic dimension (d) we evaluated our method to embed a sphere, where $s > d$ is required for isometry, sampled with noise, and initialize the algorithm on a noisy hourglass shape. We also consider the case where $s = d$ is feasible with the popular "Swiss-roll" data set where we add a square hole in the middle of the manifold.

Finally we examine data lying on a half-sphere and embed into 2 dimensions.

"Hourglass" to "sphere" illustrates how the algorithm works for $s = 3$, $d = 2$. The data $\mathbf{X}$ is sampled uniformly from a sphere of radius 1 with intrinsic dimension $d = 2$. We sample $n = 10000$ points from the sphere and add i.i.d. Gaussian noise with $\Sigma = \sigma^2/s\mathbf{I}_s$. For this artificial noise, adding dimensions beyond s has no effect except to increase σ. Since the algorithm only interacts with the data $\mathbf{X}$ in the form of pairwise distances, adding noise dimensions simply adds noise to the distance and increase the computational cost of computing the distance matrix.

Estimating the Laplacian $\boldsymbol{\mathcal{L}}$ on the noisy data $\mathbf{X}$, we initialize with a noisy "hourglass" shape in $s = 3$ dimensions, with the same noise distribution as the sphere. If the algorithm works correctly, by using solely the Laplacian and weights from $\mathbf{X}$, it should morph the hourglass $\mathbf{Y}^0$ back into a sphere. The results after convergence at 400 iterations are shown in Fig. 3.1 We see that RR not only recovers the sphere, but it also suppresses the noise. In this case it only RRaccepts parameters s and d such that $s > d$ and therefore we cannot compare with other algorithms.

"Swiss Roll with Hole" and "Hourglass" to "sphere" compare RR to several embedding algorithms with respect to geometric recovery. The algorithms are Isomap, Laplacian Eigenmaps, HLLE[18], MVU [1] which were described in chapter 2. . The embeddings $\mathbf{Y}^{LE,MVU,HLLE}$ need to be re-scaled before being evaluated, and we use a Procrustes transformation to the original data. The algorithms are compared with respect to the co-metric distortion Loss, and with respect to mean squared error in pairwise distance. That is we use the following distortion measure:

$$\mathrm{dis}(\mathbf{Y}, \mathbf{Y}^{true}) = {}^{2}\!/_{n(n-1)} \sum_{k \neq k'} \left(||\mathbf{Y}_k - \mathbf{Y}_{k'}|| - ||\mathbf{Y}^{true}_k - \mathbf{Y}^{true}_{k'}|| \right)^2 \tag{3.16}$$

where $\mathbf{Y}$ is the embedding resulting from the chosen method and $\mathbf{Y}^{true}$ are the true noiseless coordinates. The loss optimized by Isomap can be seen as an estimate of this pairwise

[1] These embeddings were computed using drtoolbox: https://lvdmaaten.github.io/drtoolbox/ [54]

distortion, however they estimates the distance between points on in $\mathbf{X}$ on the manifold $\mathcal{M}$ using the graph shortest path algorithm, here we replace these with the true distances. Note that none of Isomap, MVU, HLLE could have been tested on the hourglass to sphere data of the previous example, because they work only for $s = d$. The sample size is $n = 3000$ in both experiments, and noise is added as described above for "hourglass" to "sphere".

Flat "Swiss roll" manifold, $s = d = 2$. The results are displayed in Fig. 3.3. In this case the manifold can be embedded into $\mathbb{R}^d$ isometrically and so all algorithms ought to have their assumptions met. In this case we have a ground-truth isometric embedding to which we can evaluate each method.

Curved "half sphere" manifold, $s = d = 2$. In this case, an isometric embedding into 2 dimensions is not possible. We examine which of the algorithms achieves the smallest distortions in this scenario. The true distances were computed as arc-lengths on the half-sphere. We note that no algorithm can feasibly achieve 0 error. The results are displayed in Fig 3.3.

Since RRis iterative and optimizes a non-convex loss function, in order to evaluate how it performs we initialized RRwith the output of each method to which we are comparing. For almost every initialization and noise level, RR achieves a decrease in dis, in some cases significant decreases. Isomap also performs well and even though RR optimizes a different loss function it never increases dis and often improves on it. This demonstrates the ability of the Riemannian Metric to encode simultaneously all aspects of manifold geometry. Convergence of RR varies with the initialization but was in all cases faster than Isomap which has the problem of performing an eigendecomposition of a dense $n \times n$ matrix, something which does not scale well. On the other hand, the extension of RR to PCS-RR allows for scaling to much larger data sets, which we demonstrate in the next section.

3.4.2 Visualizing the main SDSS galaxy sample in spectra space

In chapter 1 we discussed the large data sets that are now being recorded in the field of astronomy; in this section we describe an application of manifold learning to these data.

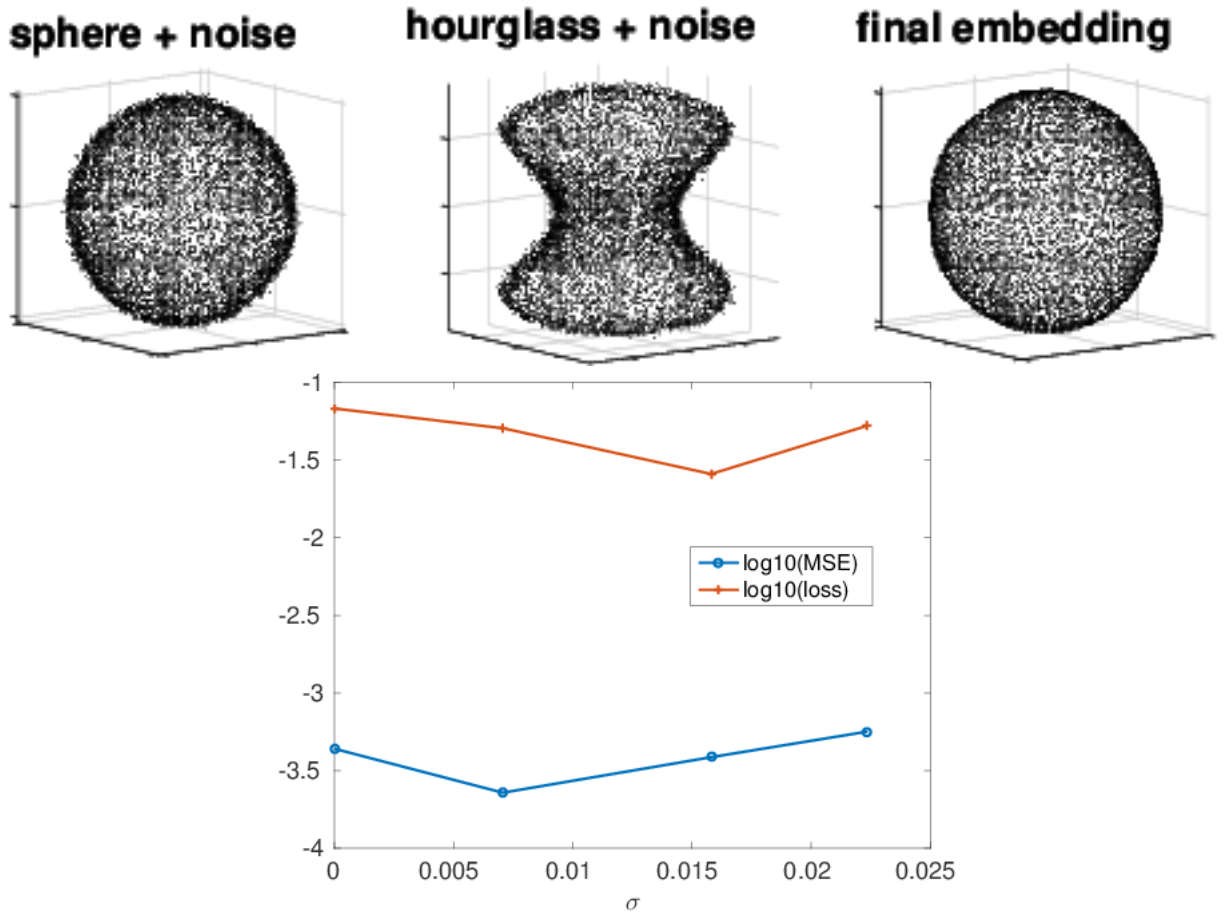

Figure 3.1: **Hourglass to sphere.** From left to right: target $\mathbf{Y}$ (noisy sphere), initialization $\mathbf{Y}^0$ of RR (noisy hourglass), output of RR, mean-squared error and Loss vs. noise level σ (on a $\log_{10}$ scale). Convergence of RR was achieved after 400 iterations.

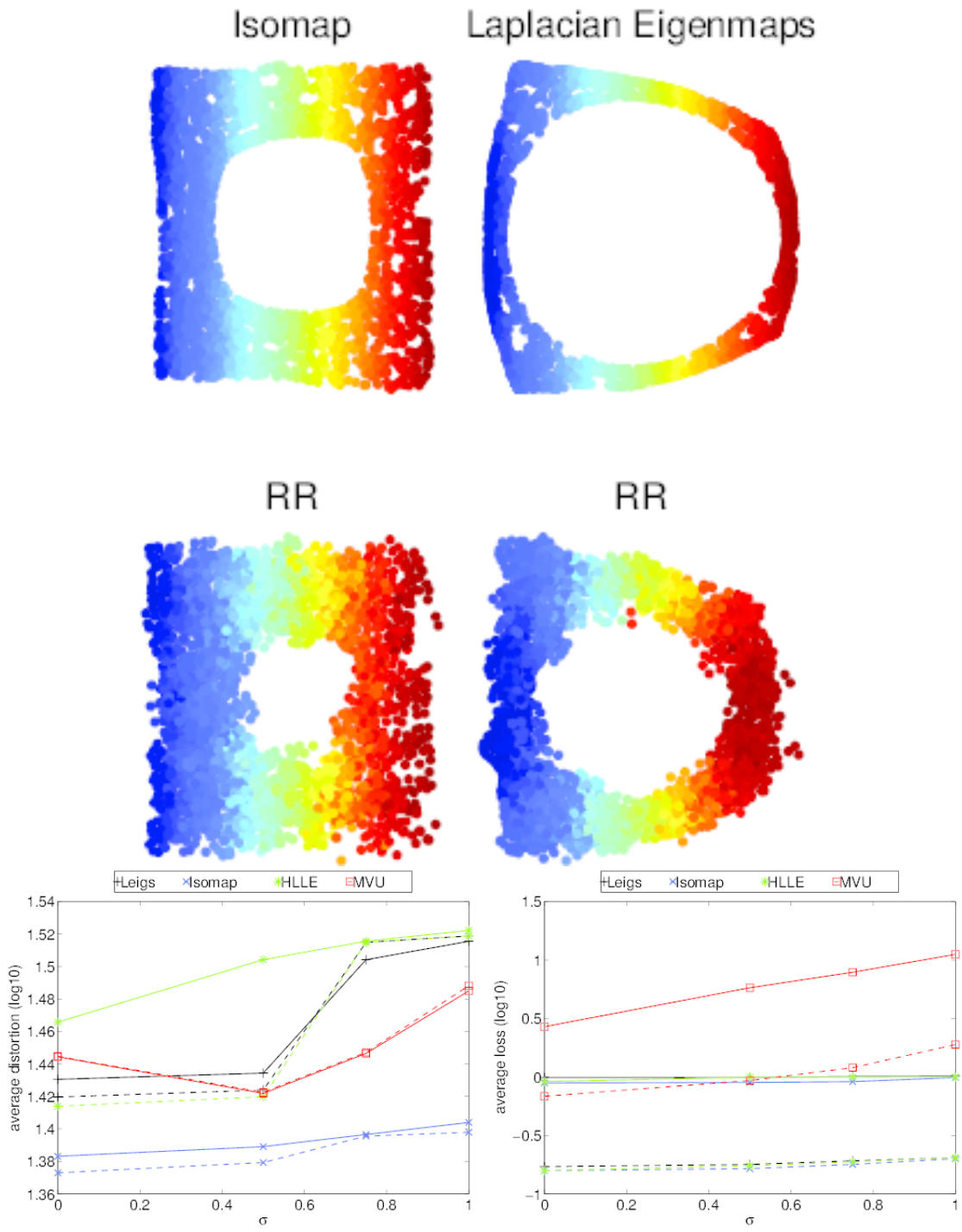

Figure 3.2: Swiss hole Top plots display example initial embeddings, middle row displays the Riemannian Relaxed versions. The bottom row displays dis value vs. noise level σ on the left and Loss value vs. noise level σ on the right.As RR was initialized at each method dashed lines indicated relaxed embeddings

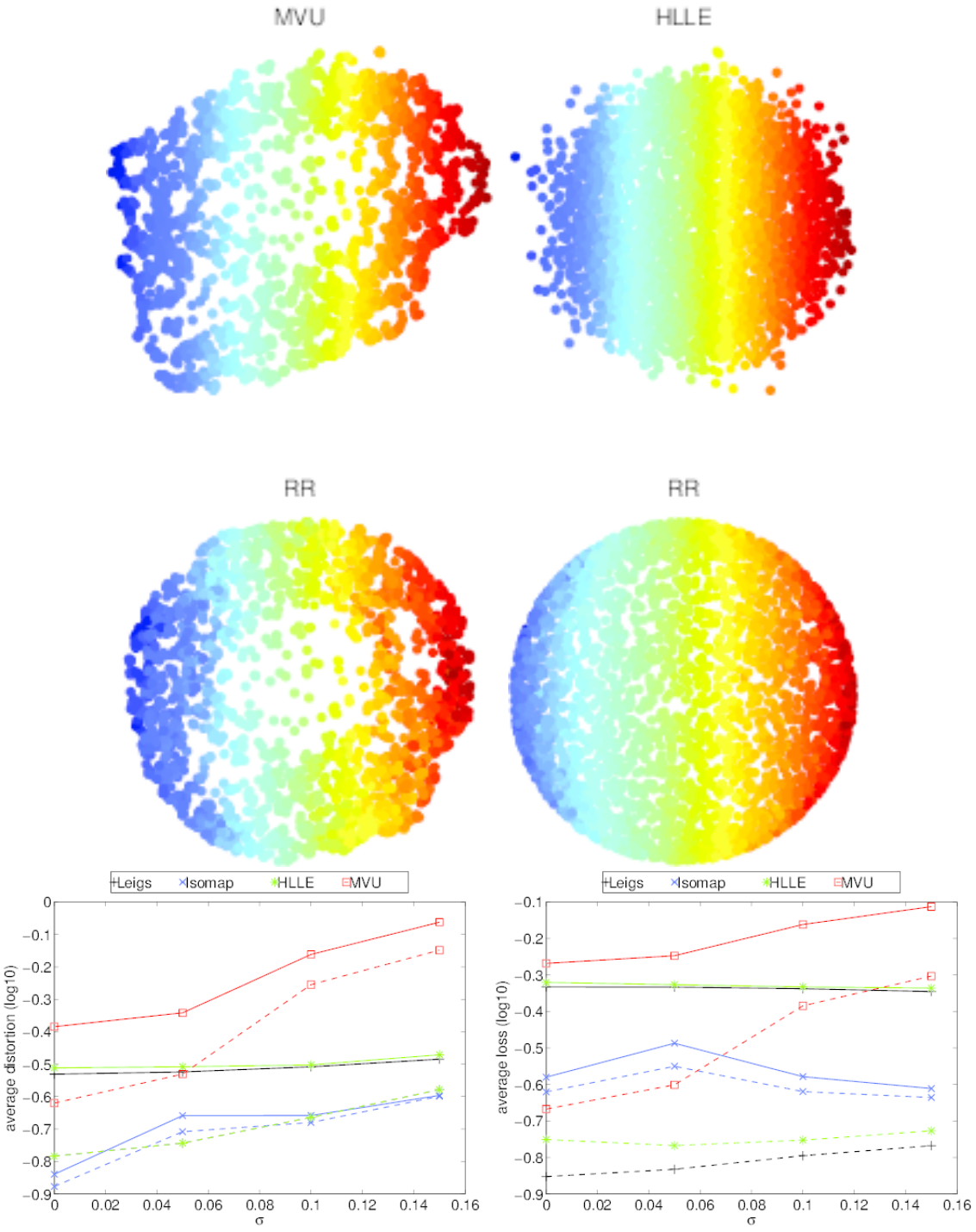

Figure 3.3: half sphere (right). Top plots display example initial embeddings, middle row displays the Riemannian Relaxed versions. The bottom row displays dis value vs. noise level σ on the left and Loss value vs. noise level σ on the right.As RR was initialized at each method dashed lines indicated relaxed embeddings

The data consists of spectra of galaxies from the Sloan Digital Sky Survey [1] as described in chapter 1. A log-log plot of the average number neighbors $m(r)$ vs. neighborhood radius r (shown in 4.1), indicates that the intrinsic dimension of these data varies with the scale r. In particular, in order to support $m = O(d)$ neighbors, the radius must be at least 60, in which case $d \leq 3$. See chapter 4 for a longer discussion on how the dimension and radius were chosen for these data. We ultimately chose to embed into $s = 3$ dimensions with a radius of 66. We embedded the whole data set by Laplacian Eigenmaps, obtaining the graph in Fig. 3.4. This figure strongly suggests that d is not constant for this data cloud, and that the embedding is not isometric (Fig 3.6).

In order to analyze the data at different intrinsic dimensions we "re-scaled" the data along the three evident principal curves shown in Figure 3.5 by running PCS-RR ($\mathbf{Y}, n = 10^5, n' = 2000, s = 3, d = 1$). In the new coordinates (Fig 3.7), $\mathbf{Y}$ is now close to isometric along the selected curves, while in Fig. 3.6, $||\mathbf{H}_k||$ was in the thousands on the uppermost "arm". This means that, at the largest scale, the units of distance in the space of galaxy spectra are being preserved (almost) uniformly along the sequences, and that they correspond to the distances in the original $D = 3750$ data. Moreover, we expect the distances along the final embedding to be closer on average to the true distance, because of the de-noising effect of the embedding. In 3.8 we show the embedding after RR re-scaling where the data are colored by Hydrogen-α emission which is an example of a non-linear feature that would not have been preserved using a linear dimension reduction technique.

3.5 Related Work & Discussion

In this chapter we propose a new, natural, way to measure the distortion from isometry of any embedding $Y \in \mathbb{R}^{n \times s}$ of a data set $X \in \mathbb{R}^{n \times D}$, and study its properties. The loss we propose is mathematically motivated by the definition of an isometric embedding into Euclidean space of a d dimensional sub-manifold of $\mathbb{R}^D$ into $\mathbb{R}^s$. The distortion loss is based on an estimate of the pushforward Riemannian metric into Euclidean space $\mathbb{R}^s$ and the optimal embedding is found by iteratively updating the embedding to minimize the distortion of the

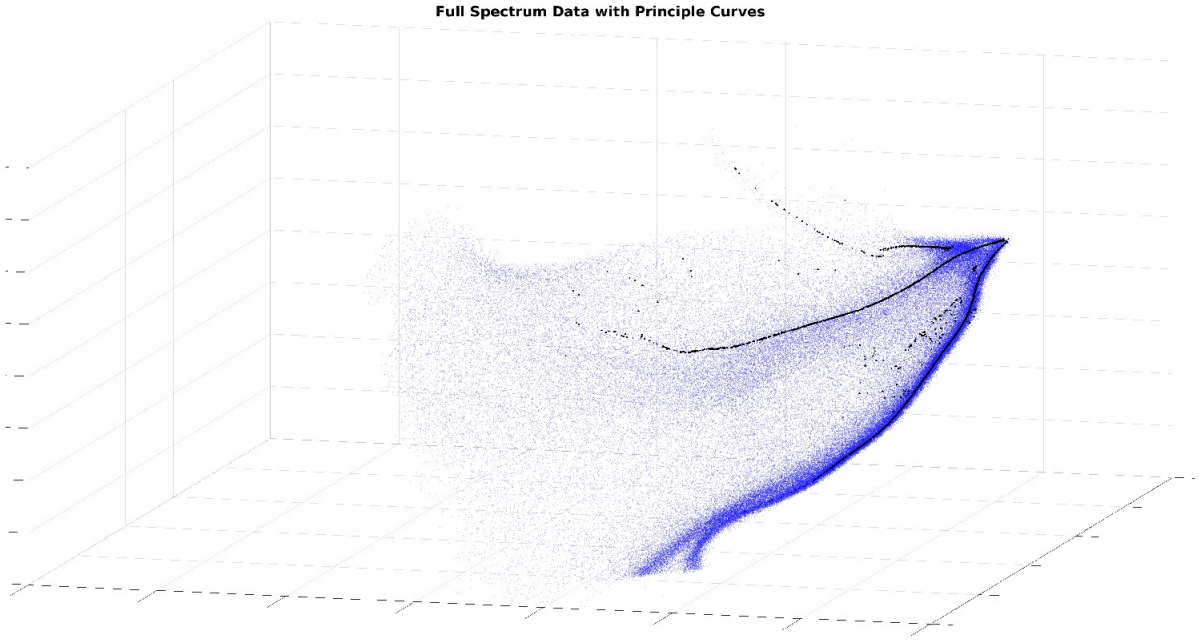

Figure 3.4: Initial LE embedding from $D = 3750$ to $s = 3$ dimensions. After viewing the initial embedding we observed that the dimension d appeared to vary along the manifold with a distinct set of "arms" and a thin "veil" between. We identify the arms with the principal curves $\hat{\mathbf{Y}}$ which are superimposed in black.

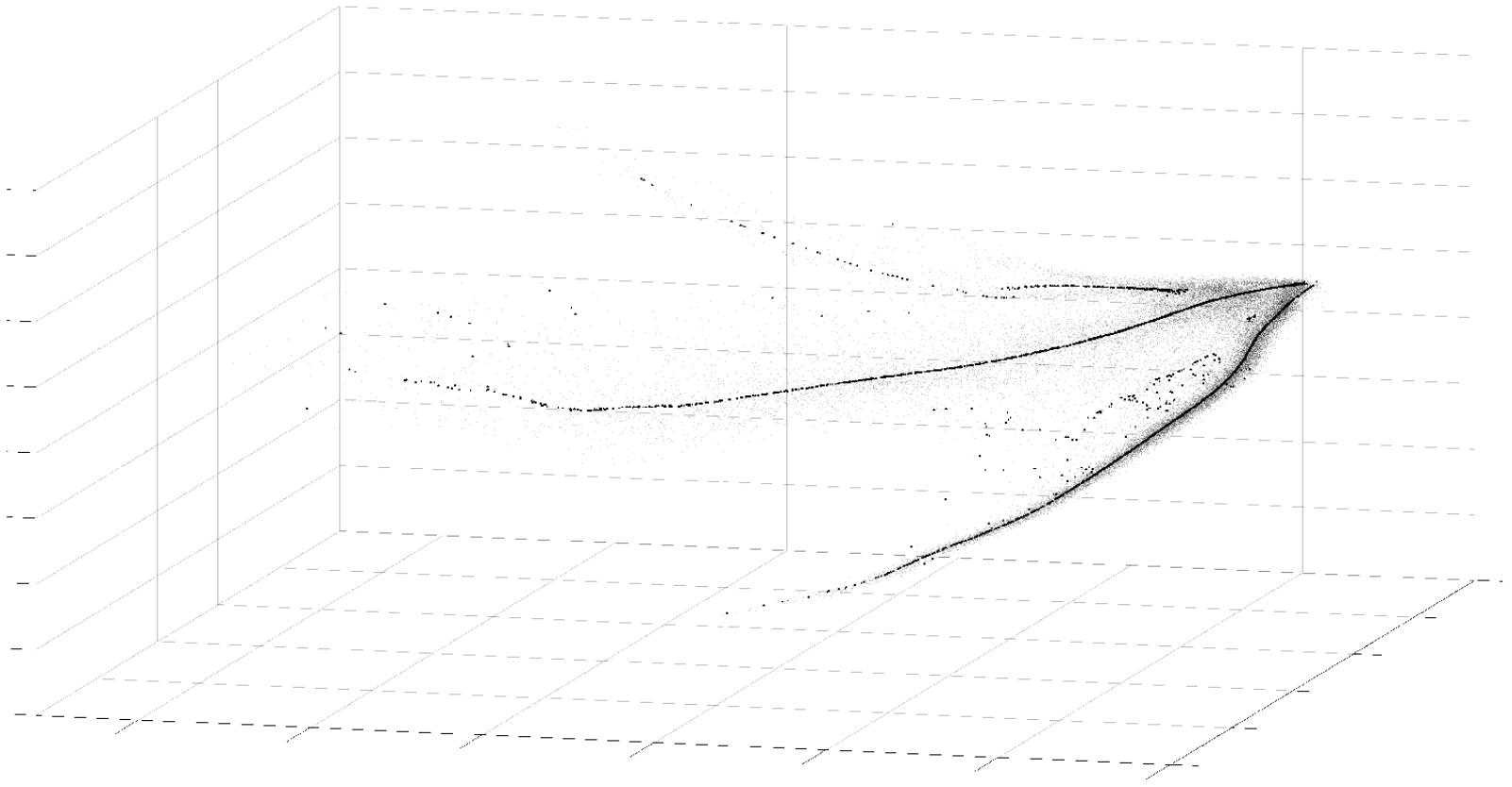

Figure 3.5: **a:** Initial LE embedding from $D = 3750$ to $s = 3$ dimensions, with the principal curves $\hat{\mathbf{Y}}$ superimposed. For clarity, we only show a small subsample of the $\mathbf{Y}^0$

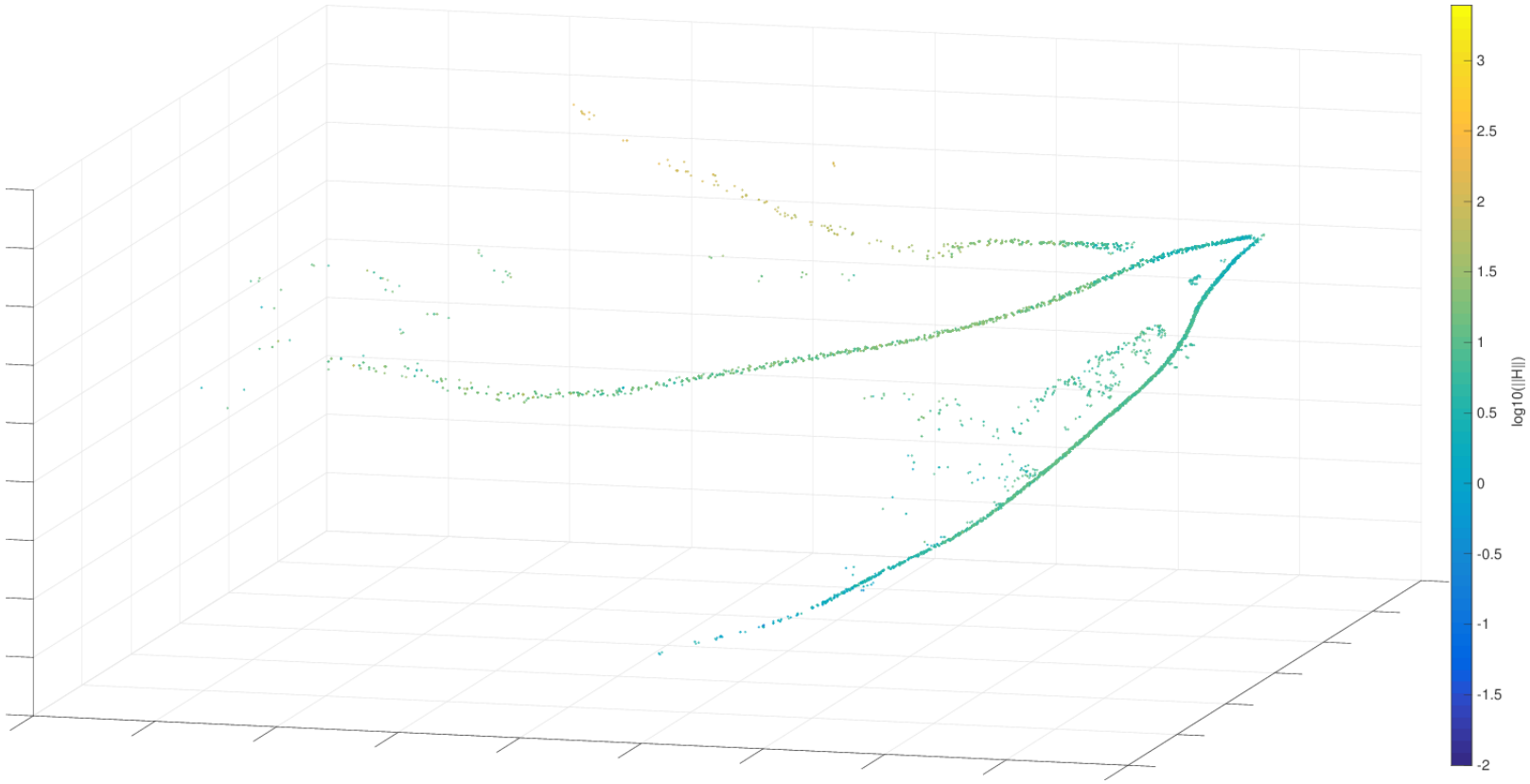

Figure 3.6: **b:** same embedding as **a, only points "on" principal curves, colored by** $\log_{10}||\mathbf{H}_k||$ **(hence, 0 represents isometry)**

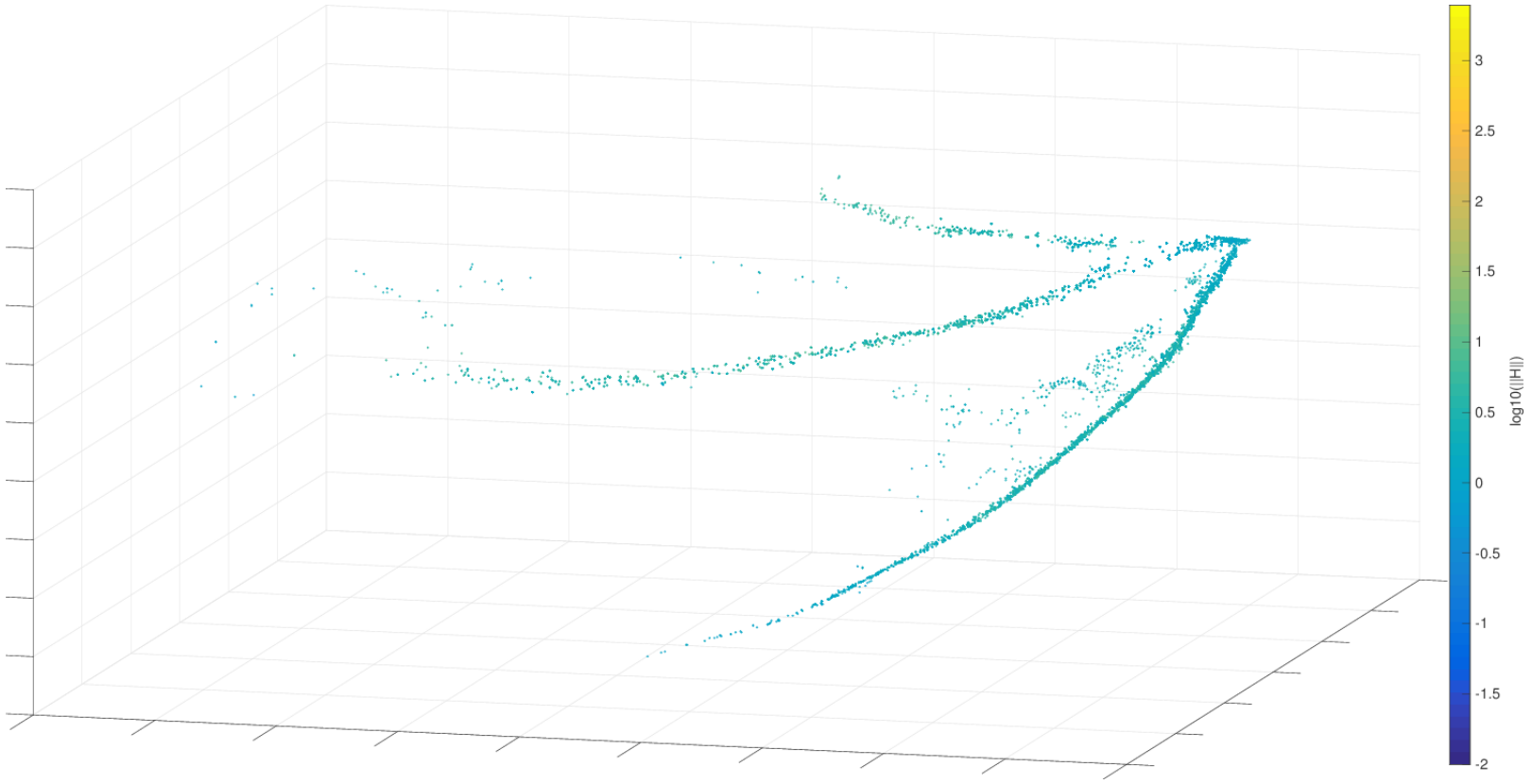

Figure 3.7: **c:** same points as in (b), after RR(color on the same scale as in (b))

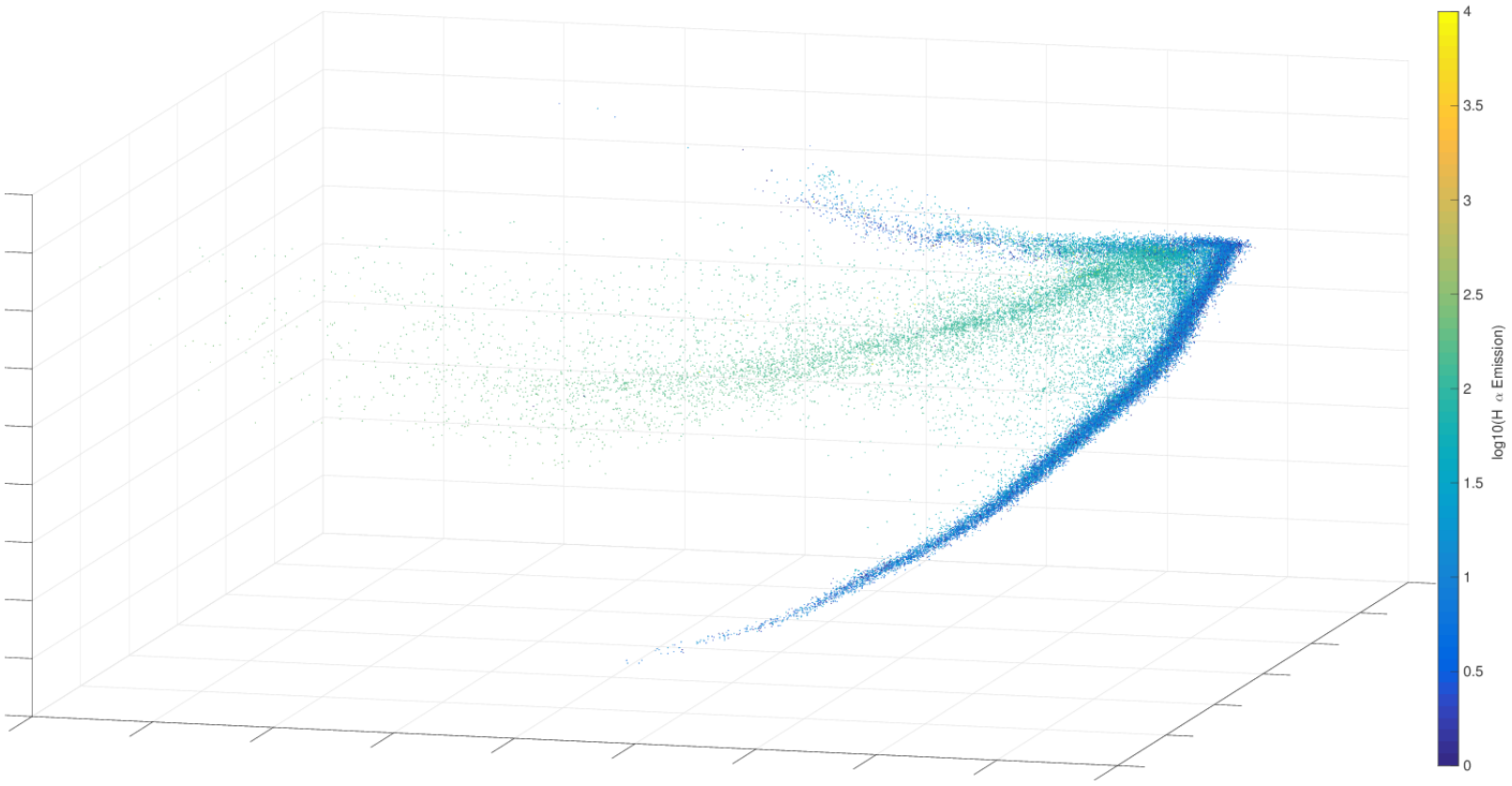

Figure 3.8: **d:** 40,000 galaxies in the coordinates from (c), colored by the strength of Hydrogen α emission, a very nonlinear feature which requires dozens of dimensions to be captured in a linear embedding. Convergence of PCS-RR was achieved after 1000 iterations and took 2.5 hours optimizing a Loss with $n' = 2000$ terms over the $n \times s = 10^5 \times 3$ coordinates, corresponding to the highest density points.

embedding into Euclidean space.

The algorithm, RR ,we propose departs from many existing non-linear embedding algorithms in several ways. First, instead of a heuristically chosen loss, like pairwise distances, or local linear reconstruction error, it directly optimizes the Riemannian co-metric of the embedding $\mathbf{Y}$. When this is successful, and the loss is 0 *all* geometric properties (lengths, angles, volumes) are preserved simultaneously. From the computational point of view, the non-convex loss is optimized iteratively by projected gradient this has many advantages including scaling to large data, similar to t-SNE. Nevertheless, the loss we define directly measures the deviation from isometry, unlike t-SNE which involves minimizing a K-L divergence; therefore, when the loss of Riemannian Relaxation is (near) 0, (near) isometry is achieved.

Third, our algorithm explicitly requires both an embedding dimension s and an intrinsic dimension d as inputs. Most existing embedding algorithms, as Isomap, LLE, HLLE, MVU, LTSA only work in the case $s = d$, while Laplacian Eigenmaps/Diffusion Maps requires only s but does not attempt to preserve geometric relations (nor ensure that the resulting data is still a manifold of dimension d. Our algorithm therefore departs significantly in that it attempts to find an isometry from one sub-manifold of Euclidean space to another sub-manifold of Euclidean space but where the latter is of significantly smaller ambient dimension but explicitly the same *intrinsic* dimension. Finally, RR is computationally competitive with existing algorithms, and can be seamlessly adapted to a variety of situations arising in the analysis of real data sets.

Chapter 4

PARAMETER SELECTION AND OUT-OF-SAMPLE EXTENSION

The greater goal of this thesis when all its chapters are taken together, is to provide a "pipeline" for performing manifold learning. There are many paths along the way and decisions which must be made and we strive to provide for the researcher a guide in making these decisions. In chapter 3 we described the goals of manifold learning in detail as well as proposed a novel method for finding an isometric embedding of the data. In Riemannian Relaxation and almost all other manifold learning algorithms there are a set of parameters which are taken as given in most algorithms. Their selection, however, cannot be trivialized as they have great impact on the resulting embedding. The parameters are: the **neighborhood bandwidth** h, the embedding dimension s, and the **intrinsic dimension** d. Incorrect radius selection can lead to disconnected results (when the bandwidth is too small) or incorrect results (when the bandwidth is too large).

Incorrect determination of the intrinsic dimension can create geometric restrictions that can be impossible to meet, for example attempting to embed a d-sphere into d dimensions. We know from Whitney and Nash as in §2.2 that seeking an isometry into Euclidean space usually requires that $s > d$. Additionally, there is often a relationship between these variables. Changing the bandwidth parameter h changes the size of the neighborhoods and the measured similarity between points. As the radius increases the data appear to be more equidistant as even large distances become negligible compared to h. So that as the bandwidth is varied the dimension might also appear to vary. Consequently there is a back-and-forth in estimating these parameters and so often tasks must be iterated or performed over a variety of values. In §4.1 we describe radius estimation, and in §4.2 we describe intrinsic dimension estimation.

There are many different heuristics which have been used to estimate parameters such as the neighborhood bandwidth and the intrinsic dimension[11, 12, 28, 29]. In this chapter we describe some of these heuristics but more importantly we describe how we have combined them together to make a cohesive and sensible method for estimating these values with the specific goal of manifold learning in mind. In addition to parameter selection we provide an **out-of-sample extension** tool via the Nyström Method in the §4.3.

4.1 Bandwidth Estimation

There are many proposed ways to estimate an optimal bandwidth parameter for performing manifold learning. Due to its relationship with the Riemannian Metric we consider the method proposed by [43]. This method, however, taken at as written for example in algorithm 4, does not scale well enough to apply to large data sets. Therefore in this chapter we present adaptations to the algorithm of [43] which allows for scaling to large data sets. Finally we motivate this need for scaling with an application to the astronomy data where we use this method to determine the optimal bandwidth parameter.

We briefly summarize the method of [43]. We are interested in estimating the bandwidth, h of the of the (truncated) Gaussian Kernel $K_h(z) = \exp(-z^2/2h^2)$, $|z| < rh$ where r is a truncation constant, since 99.9% of the mass of a Gaussian random varaible is within 3 standard deviations of 0 using $r = 3$ is common and what is done throughout this chapter. This is used in computing the Laplacian matrix and consequently the Riemannian Metric as described in chapter 3. In particular we optimize the bandwidth with respect to the loss in (3.1) based on estimating the tangent space at each point. If the data lie on a manifold of intrinsic dimension d then the (local) chart $df : \mathcal{M} \to \mathcal{T}_p(M)$ is a trivial isometry at p and so the deviation from isometry as measured by (3.1) should be small, any deviation from zero must be a consequence of using finite data approximations. Therefore we select a bandwidth which minimizes this distortion. We use Weighted Principal Component Analysis to estimate $T_p(M)$ then compute loss of (3.1), $\text{Loss}(df(p))$, evaluated at each point. We do this for a sequence of bandwidths and choose the one which has minimal loss. The computation is summarized in algorithm 4.

Naively implemented algorithm 4 will not scale to large data sets. With a few small observations, however, it is possible to apply this algorithm to the astronomy data of size $N = 675,000$ and $D = 3750$. The two important observations are that:

1. Most of the operations performed are *local operations*

Input : Data matrix $\mathbf{X} \in \mathbb{R}^{n \times D}$, vector bandwidths $\mathbf{h}$, dimension d'

Output: Optimal bandwidth h

init : $\mathbf{l} = \mathbf{0}$

for *each bandwidth* h *in* $\mathbf{h}$ **do**

  $l \leftarrow 0$;

  Compute weights $\mathbf{W} \leftarrow$ by equation 2.3;

  Compute Laplacian $\boldsymbol{\mathcal{L}} \leftarrow$ by equation 2.4;

  for *each point* i *in* $\boldsymbol{X}$ **do**

    Estimate the Local Tangent space $\mathbf{Y}$ by Weighted PCA with weights $\mathbf{W}_i$, data $\mathbf{X}$, and into dimension d';

    Add distortion to total distortion: $l+ = \leftarrow Loss(\boldsymbol{\mathcal{L}}, \mathbf{Y})$ by equation (3.1);

  end

  $\mathbf{l}[r] \leftarrow l/n$;

end

Set $h \leftarrow \arg\min_h l[h]$;

Algorithm 4: `Estimate Bandwidth - Perrault-Joncas & Meilă`

2. Many operations can be performed in parallel

In more detail we make the following observations. When computing the Riemannian Metric H we note that it is a *locally* estimated quantity, in that H only depends on $\boldsymbol{\mathcal{L}}$ through the row $\boldsymbol{\mathcal{L}}_i$.

Definition 19. *(Neighborhood) Let the neighborhood of i be defined as:*

$$nbh_h(i) = \{j : ||X_i - X_j||^2 < 3h\} \tag{4.1}$$

For a subset of points I let $\mathrm{nbh}_h(I) = \cup_{i \in I} \mathrm{nbh}_h(i)$.

Definition 20. *(Greater Neighborhood) Let the greater neighborhood of i be defined as:*

$$gnbh_h(i) = nbh_r(nbh_h(i)) \tag{4.2}$$

The row of $\boldsymbol{\mathcal{L}}$ corresponding to i only depends on the (greater) neighbors of point i. Therefore we only need to compute rows of the affinity matrix for the points in $\text{gnbh}_r(i)$ Similarly since the Riemannian metric H only depends on the row of $\boldsymbol{\mathcal{L}}$ for point i it also only depends on the embedding $\mathbf{Y}$ through the rows of $\mathbf{Y}$ corresponding to $\text{nbh}_r(i)$. Therefore we only need to project the points $\text{gnbh}_r(i)$ onto the tangent space – this alone is a significant speedup since naively this it is a dense multiplication of an $(n \times D)$ matrix with a $d \times d$ matrix.

The next observation is that only a subset of the points need to be evaluated. If a data set that is large enough such that for points i and j we have that $nhb_h(i)$ does not differ significantly from $nbh_h(i)$ (that is they both contain mostly the same points) then $H_i \approx H_j$ for $j \in \text{nbh}_r(i)$ since this would mean $X_i \approx X_j$ and they rows of $\boldsymbol{\mathcal{L}}$ will be similar since the have similar neighborhoods.

Additionally, if we assume $\mathcal{M}$ is smooth and then recall that the loss in (3.1) is the discrete estimate of an integral on $\mathcal{M}$ then by sub-sampling we are replacing this with a Monte-Carlo estimate of the integral. Hence, we don't need to evaluate the loss function on *all* data points. These remarks are implemented in algorithm 5.

If there are multiple cores available, however, then the algorithm can be sped-up even further by taking advantage of parallelization. The algorithm can parallelize either across data points or across across bandwidths. If parallelizing across data points it is advantageous to initialize the eigen-decomposition for bandwidth h_{t+1} with the value in h_t (assuming they are sorted) this can both stabilize any randomly seeded eigendecomposition algorithm the computation as well as speed up any iterative algorithm by improving the initialization. However, since generally the sample size will be larger than the number of bandwidths to test and significantly larger than the number of available cores it's advantageous to parallelize across bandwidths. Additionally, if shared memory available then a single copy of $\mathbf{X}$ and the distance matrix can be stored in shared memory and then copies of $\text{nbh}_r(i)$ can be distributed

to local memory. Finally, only

$$S(I) = \bigcup_{i \in I} \text{gnbh}_r(i) \tag{4.3}$$

where I is the subset of points over which the data will be evaluated, subset of the data need to be stored in memory during the procedure.

To demonstrate its ability to handle large data we show an example of using 5 on the astronomy data described in chapter 3 and chapter 5 in figure 4.1. We chose to parallelize over bandwidths since the disadvantage of parallelizing over bandwidths is that we may lose stability in the eigendecomposition phase (as we do not have a strong pre-conditioner). As discussed in [43] the algorithm need not be evaluated at the true intrinsic dimension d of the manifold but rather for any $d' \leq d$. In fact it was demonstrated that using small d' increased stability and accuracy in synthetic examples. Since we are often evaluating small dimensions $d' < 5$ we feel this disadvantage is outweighed by the ability to dedicate a processor to each bandwidth. In particular in the case of $d' = 1$ when computing the loss of 3.1 the norm reduces to $|H - 1|$ where H is the estimated Riemannian pushforward co-metric, in this case there is no advantage to storing the value at bandwidth h_{t-1} since it does not make computing $|H - 1|$ any faster or more stable. Therefore we choose to parallelize over bandwidths.

4.2 Intrinsic Dimension Estimation

Estimating intrinsic dimension, d, is challenging. Here we use the doubling dimension. We know from [15] that for a d-dimensional sub-manifold of R^D is the doubling dimension k is $\mathcal{O}(d)$.

Definition 21. *(Doubling Dimension) The doubling dimension is the smallest integer k such that any ball of radius r of the space X, $B_r(X)$ can be covered by 2^k balls of radius $\frac{r}{2}$.*

We can use these two facts to generate a plot which will help us determine a good estimated dimension. Since we are dealing with a data set instead of a manifold we will deal

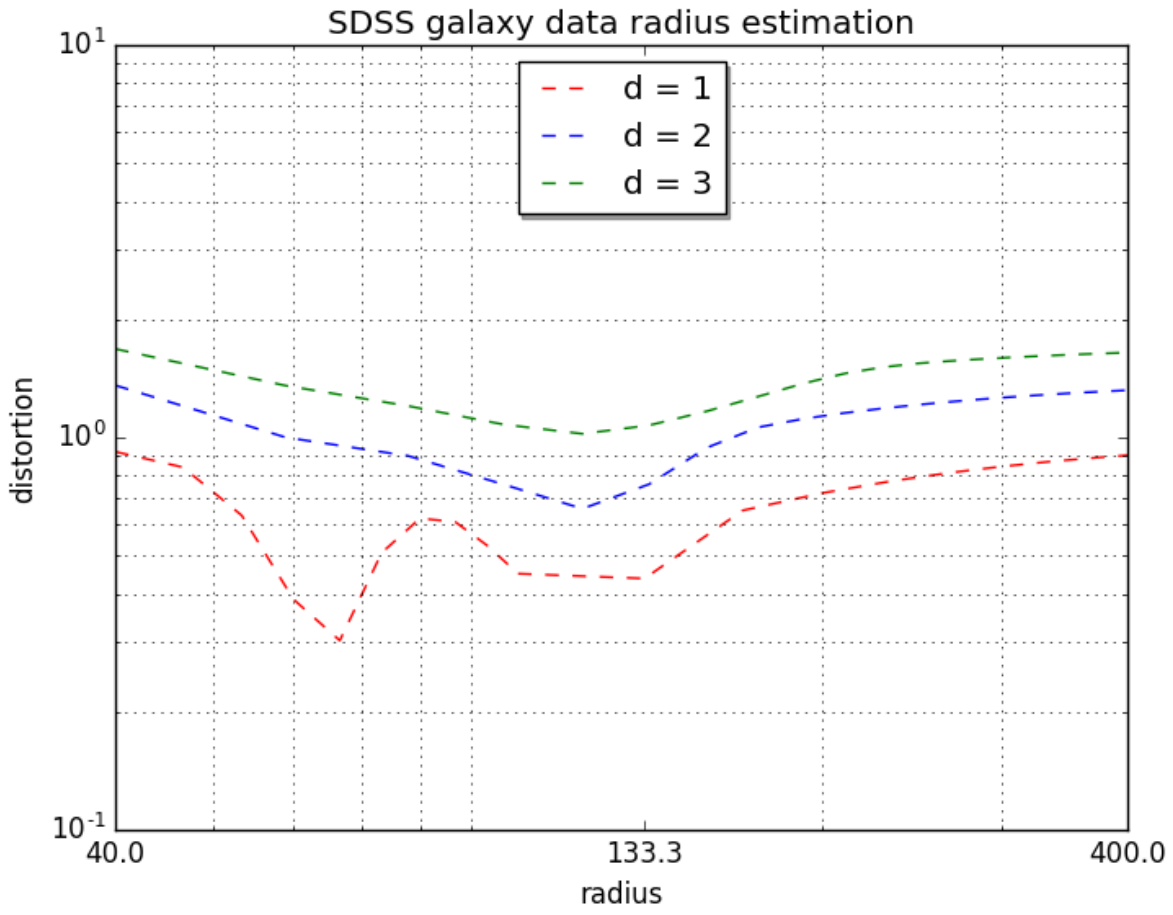

Figure 4.1: In this figure we apply the bandwidth estimation technique as described in algorithm 5. The SDSS data consists of $n = 650,000$ data points observed in $D = 3750$ dimensions. Here we take a sample of size 200 points taken from the principal curves as discussed in chapter 3. We evaluate the bandwidths in parallel for three different values of the tangent plane dimension d. We take the "radius" to be $r = 3h$. We note that, in confirming with the experiments in [43] that smaller d results in more stable results. Based on this estimation we know that a good bandwidth is between 50/3 and 100/3. We chose a radius of $h = 66/3$ for these data which was the minimum for $d' = 1$.

Input : Data matrix $\mathbf{X} \in \mathbb{R}^{n \times D}$, vector bandwidths $\mathbf{h}$, dimension d', subset index I

Output: Optimal bandwidth h

Init : $\mathbf{l} = \mathbf{0}$

Init : $S \leftarrow S(I)$ from (4.3)

Init : Read in subset of data as $\mathbf{X}_s \leftarrow \mathbf{X}[S, :]$

for *each bandwidth* h *in* $\mathbf{h}$ *asynchronously in parallel* **do**

  $l \leftarrow 0$;

  Compute weights $\mathbf{W} \leftarrow$ by equation 2.3;

  Compute Laplacian $\boldsymbol{\mathcal{L}} \leftarrow$ by equation 2.4;

  for *each point* i *in* I **do**

    $\mathbf{X}_i \leftarrow \mathbf{X}[\mathbf{nbh}(i), :]$;

    Estimate the Local Tangent space $\mathbf{Y}_i$ by Weighted PCA with weights $\mathbf{W}_i$, data $\mathbf{X}_i$, and into dimension d';

    Add distortion to total distortion: $l+ = \leftarrow Loss(\boldsymbol{\mathcal{L}}_I, \mathbf{Y}_i)$ by equation (3.1);

  end

  $\mathbf{l}[r] \leftarrow l/n$

end

Algorithm 5: `Estimate Bandwidth For Big Data`

with sets of points falling within a radius of a given point:

$$B_r(X_i) = \{X_j : ||X_i - X_j|| \leq r\}$$

Note that $|B_r(X_i)|$ is a measurement of *volume* and so ought to scale exponentially with *intrinsic* dimension. We relate this further to doubling dimension which we know scales linearly with intrinsic dimension from [15]. In particular define the *volume* of $B_r(X_i)$ to be:

$$b_i(r) \equiv |B_r(X_i)|$$

In analogy to the definition of doubling dimension, in place of the the ball being covered 2^k balls of radius $\frac{r}{2}$ we require 2^k times as many balls of radius $\frac{r}{2}$ to capture as many points as

the ball of radius r. That is:

$$b(r) = 2^k b(r/2)$$

We note that this recurrence is geometric and so the solution is:

$$b(r) = Cr^k$$

Since, $2^k b(r/2) = 2^k C(r/2)^k = 2^k(1/2)^k Cr^k = Cr^k = b(r)$. We note that this implies:

$$\log(b(r)) = \log(C) + k\log(r)$$

On the data set we estimate $b(r)$ by taking the average of $b_i(r)$ over the data points:

$$b(r) = \frac{1}{n}\sum_{i=1}^{n} b_i(r)$$

We then take a log-log plot of r versus $b(r)$. Using ordinary least squares over (potentially a subset) of this graph and taking the slope gives an estimate of the doubling dimension k. Even though the theorem of [15] only gives that $d = \mathcal{O}(k)$ experiments on spheres in figure 4.2 we demonstrate that for $d \leq 20$ we have that $d \approx k$ under the estimation algorithm described in 6. Finally, since the matrix of nearest neighbors must be calculated already (for the maximum radius to be considered) for all manifold learning algorithms this is a trivial computational step that can be done before performing an embedding.

In figure 4.2 we evaluate the performance of 6 to estimate the dimension of a sphere. We show that if we pick the region of radii to evaluate in a reasonable manner (i.e. not too large or too small as to distort the geometry or connectedness) then this method can successfully estimate the dimension. We show that for small values of d this is a fairly unbiased estimator of d. When d increases it tends to under estimate the true dimension. This is since we know that the doubling dimension is $\mathcal{O}(d)$ but may no be d exactly. Since it is often the case, in our experience, that we observe $d < 25$ this method seems to be reasonable for estimating the dimension.

In figure 4.3 we use this method to estimate the dimension on the Astronomy Data. To support $\mathcal{O}(d)$ neighbors per point we require $r > 60$ for which the plot indicates $d \leq 3$.

Input : Pairwise distance matrix $\mathbf{D} \in (n \times n)$ [1] maximum radius to consider r_{max}, minimum radius to consider r_{min}

Output: estimated dimension $\hat{d}$

Init : $\mathbf{r}$ as a sequence from r_{min} to r_{max}

for *each radius* r *in* $\mathbf{r}$*:* **do**

  Compute $b_i(r)$ for all $i = 1 : n$;

  Compute $b(r) = \frac{1}{n} b_i(r)$;

end

Estimate $\hat{d}$ from by minimizing:

$$\sum_{r \in \mathbf{r}} (\log(b(r)) - (\hat{\alpha} + \hat{d} \log(r))^2$$

via Ordinary Least Squares;

Algorithm 6: `Estimate intrinsic dimension` d

Of particular note is that the radius r corresponding to dimension 3 is between 50 and 100 and this same region was selected as optimal for these data in figure 4.1. The fact that the radius plot in figure 4.1 shows dimension varying by radius (i.e. a non-linear relationship) demonstrates the potential correspondence between dimension and radius. Consequently it is important to consider both estimates together.

This combination of heuristics, estimating radius by algorithm 5 and intrinsic dimension by algorithm 6 and most importantly interpreting both of them simultaneously, is presented here as a combined-method for selecting the important parameters in manifold learning. The addition of the scaling methods in algorithm 5 as compared to algorithm 4 and that the dimension estimation, and since algorithm 6 is a negligible computational increase that this combination of methods for parameter selection scales well to large data sets as demonstrated in figures 4.1 and 4.3.

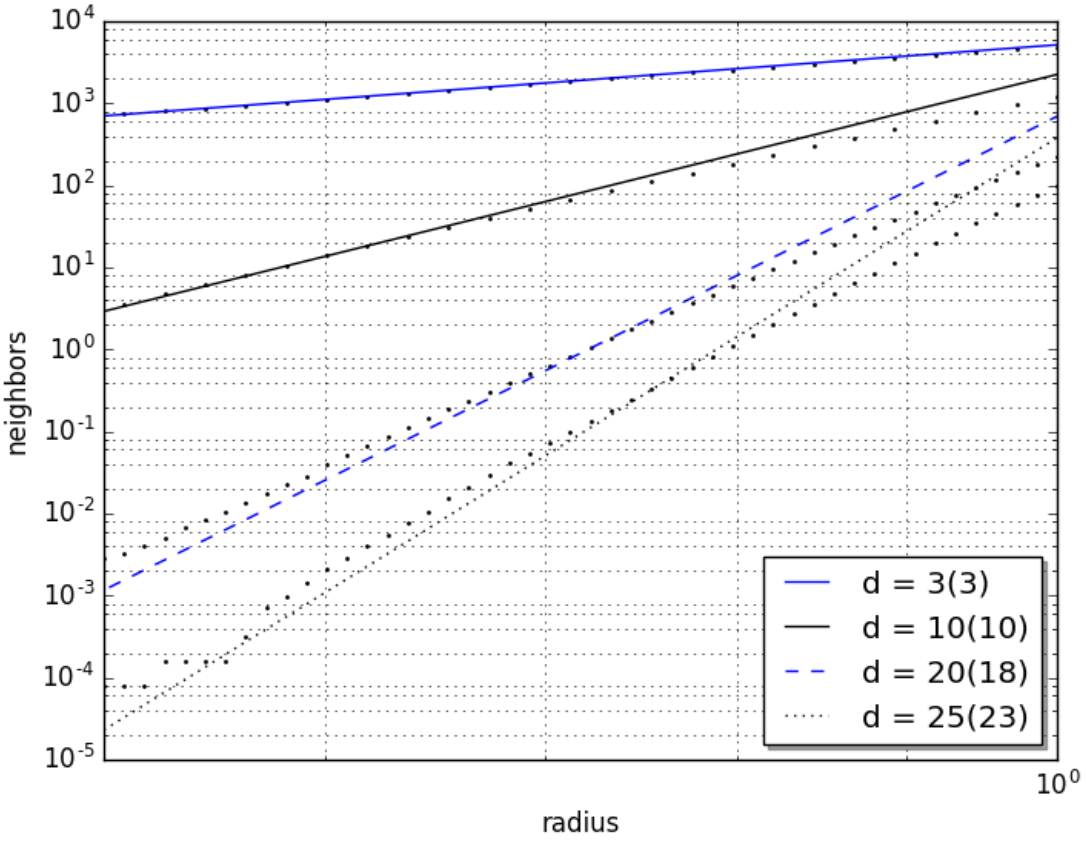

Figure 4.2: In this experiment we consider the sphere $S^d \subset \mathbb{R}^{d+1}$. We show the radius versus average number of neighbors plot for $d = 3, 10, 20, 25$. In the figure above the points denote the observed values. The lines are the best fitting lines of the true dimension d and in the parenthesis of the legend is the estimated dimension next to the true dimension.

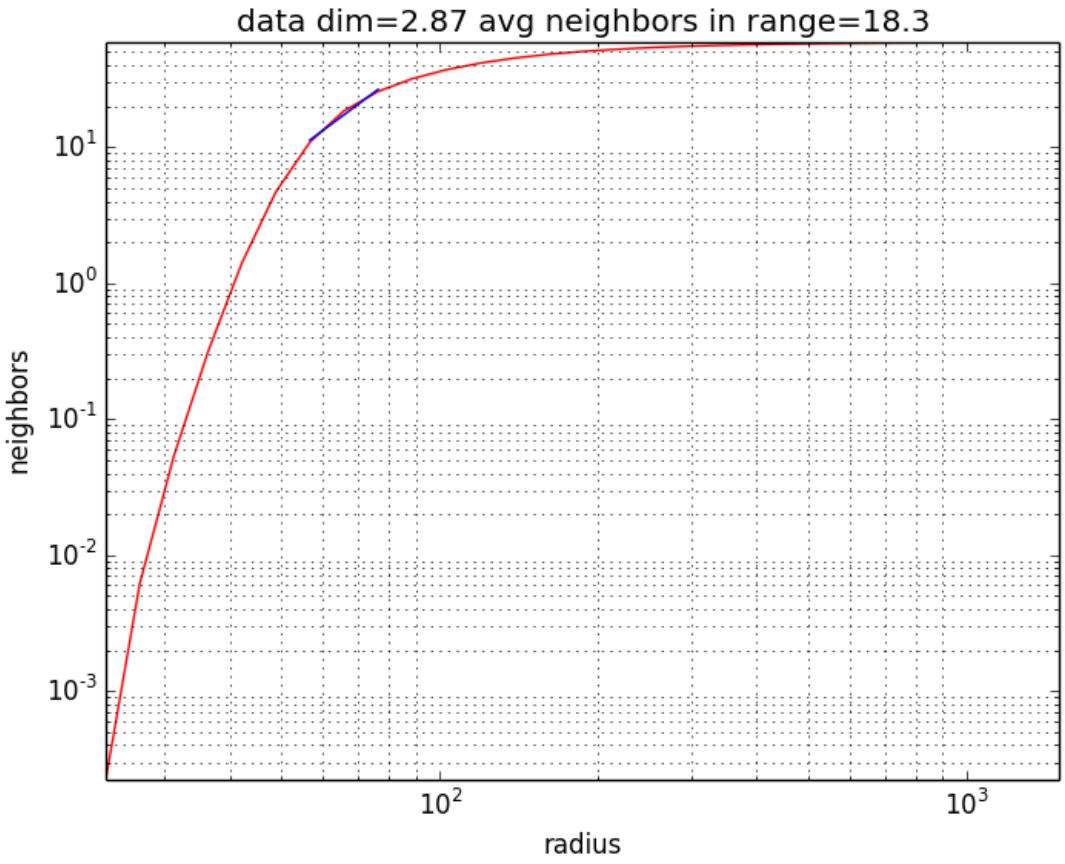

Figure 4.3: The average number of neighbors $m(r)$ vs the neighborhood radius r, on a log-log scale, for the SDSS spectra data, computed on the whole sample of 675,000 galaxies. The blue regression line, is fitted to the graph points in the shown r range, and has slope 2.87. The absence of a linear region on this graph suggests that the data dimension varies with the scale. The analysis and visualization in this chapter corresponds to the largest meaningful scale. The scale chosen, between 50 and 100 also coincides with the scale chosen in the previous section for estimating the radius based on geometric self-consistency, i.e in figure 4.1. We also note that there is some degeneracy in the neighbors estimation see chapter 5 for details.

4.3 Nyström Method for Out of Sample Extension Problem

Manifold learning is often desired as a pre-processing tool to some other task (e.g. regression or classification). In these tasks, however, we are required to estimate a model, in our case an embedding, on a *training* set and then apply that same model, i.e. the learned map ϕ, to a new data set called the *test* data. In the case of manifold learning, however, we do not learn ϕ directly in most cases. Given a set of data $\mathcal{D}_{\text{train}}$ we learn $\phi(\mathcal{D}_{\text{train}})$ rather than ϕ itself. This makes it challenging in most cases to evaluate $\phi(\mathcal{D}_{\text{test}})$ given only $\phi(\mathcal{D}_{\text{train}})$. One way to do it is to optimize over $\mathcal{D}_{\text{test}} \cup \mathcal{D}_{\text{train}}$ however this is both computationally expensive, since the stacked matrix will be larger than the both the training and test data matrices individually, and also will change the values of $\phi(\mathcal{D}_{\text{train}})$ which is undesirable. The Nyström method, however, allows for an estimation of the full eigendecompoposition of a matrix given the partial eigendecomposition of a sub-matrix and information about columns of the full matrix.

In this section we discuss one method for solving the out-of-sample extension problem for Manifold Learning. In particular since so many of the methods mentioned in this thesis are *spectral* methods, that is they involve an eigendecomposition of a matrix the Nyström method is a natural approach. The Nyström method as described in this section here has been used as a method for scaling Isomap to large data sets [50] and has been presented as a method for out-of-sample extension of manifold learning, in particular in [6]. Since we also provide an interface to the Nyström method and use it as out out-of-sample extension `predict()` function in `megaman` we present a brief overview of the Nyström method here and how it can be used to solve the out-of-sample extension problem.

In general there is no natural solution to this extension problem for manifold learning. The reason why can be summarized as follows. There are three general steps in manifold learning algorithms. The first and last step are common among all methods and the second step generally differentiates them.

1. First create a *sparse* matrix $\mathbf{D}$ using $\mathbf{X}$

- $\mathbf{D}$ is an $n \times n$ *symmetric* matrix
- $\mathbf{D}_{ij} = ||\mathbf{X}_i - \mathbf{X}_j||$ if $||\mathbf{X}_i - \mathbf{X}_j|| < r$ for some radius $r > 0$ otherwise $\mathbf{D}_{ij} = 0$

2. Next create a new matrix $\mathbf{G}$

   - $\mathbf{G}$ is created using $\mathbf{D}$
   - each method creates a different $\mathbf{G}$ from $\mathbf{D}$
   - $\mathbf{G}$ is usually symmetric positive semi-definite (SPSD)
   - $\mathbf{G}$ is *not* always sparse

3. Compute the eigendecomposition of $\mathbf{G}$

   - Usually interested only in s eigenvectors of $\mathbf{G}$
   - Corresponding to the largest (or smallest) eigenvalues call that subset $\mathbf{Y}$,
   - Then $\phi(\mathbf{X}) \leftarrow \mathbf{Y}$

As explained in chapter 2 most manifold learning algorithms don't return ϕ explicitly we but rather $\phi(\mathbf{X})$. Since ϕ is unknown it's generally impossible ϕ to a new set of points unless we have additional information. Naively to extend a manifold learning algorithm to test data we would have to create the new matrix by stacking the test and training data:

$$\mathbf{X} = \begin{bmatrix} \mathbf{X}_{\text{train}} \\ \mathbf{X}_{\text{test}} \end{bmatrix}$$

and apply steps (1) - (3) to the full data set. The issue with this is that nothing we learned in the training stage is being applied in the test stage. Additionally, the $\mathbf{Y}'_{\text{train}}$, that is the output of applying the algorithm onto $\mathbf{X}$ and taking rows corresponding to data in the training set is not guaranteed to be equal to $\mathbf{Y}_{\text{train}}$, i.e. $\phi(\mathbf{X}$train which may also be undesirable.

The Nyström method allows us to approximate the s eigenvectors corresponding to the largest s eigenvalues of a symmetric positive semi-definite matrix matrix $\mathbf{G}$ given the largest s

eigenvectors of a sub-matrix of $\mathbf{G}$. The first and second steps in manifold learning algorithms can be computed without having to re-do the computations on $\mathbf{X}_{\text{train}}$ for example we only need to compute distances between $\mathbf{X}_{\text{test}}$ and itself and $\mathbf{X}_{\text{test}}$ and $\mathbf{X}_{\text{train}}$ but we don't need to compute the distances between points in $\mathbf{X}_{\text{train}}$ and themselves because we already did this. Similarly we only need to "complete" the matrix $\mathbf{G}$ in step (2) since we already have the top left entry. Therefore, the computational challenge lies in evaluating the eigendecomposition.

Suppose that our training data has l points and our test data has k points and we have $n = k + l$ total points so that $\mathbf{X}_{train}$ is $(l \times D)$, $\mathbf{X}_{test}$ is $(k \times D)$ and $\mathbf{X}$ is $(n \times D)$. Suppose we compute our $\mathbf{G}$ matrix using $\mathbf{X}_{\text{train}}$ call it $\mathbf{G}_{\text{train}}$. The *full* $\mathbf{G}$ matrix using all the data $\mathbf{X}$ can be broken down into pieces:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{\text{train}} & \mathbf{G}'_{21} \\ \mathbf{G}_{21} & \mathbf{G}_{22} \end{bmatrix}$$

$\mathbf{G}_{\text{train}}$ is known (since it was compute during training period) but the other two matrices are unknown and have to be computed. In order to apply the Nyström extension we will need to compute compute $\mathbf{G}_{21}$. Constructing $\mathbf{G}_{21}$ will depend on the manifold learning algorithm selected, for example it may involve completing a Laplacian matrix as in Laplacian Eigenmaps or a graph-shortest-path matrix in the case of Isomap.

During training period the (partial) eigendecomposition of $\mathbf{G}_{\text{train}}$ is computed i.e if

$$\mathbf{G}_{\text{train}} = \mathbf{U}_{\text{train}} \mathbf{\Sigma}_{\text{train}} \mathbf{U}'_{\text{train}}$$

Where $\mathbf{\Sigma}_{\text{train}}$ contains the eigenvalues of $\mathbf{G}_{\text{train}}$ and $\mathbf{U}_{\text{train}}$ the corresponding eigenvectors. In general we don't compute the full eigendecomposition but rather we find $\mathbf{\Sigma}_{\text{train},s}$ and $\mathbf{U}_{train,s}$ where $\mathbf{\Sigma}_{\text{train},s}$ contains the s largest (sometimes smallest) eigenvalues of $\mathbf{G}_{\text{train}}$ and $\mathbf{U}_{\text{train},s}$ contains the corresponding eigenvectors. Generally $\phi_{\text{train}}(\mathbf{X}_{\text{train}}) = \mathbf{U}_{\text{train},s}$.

If we want to find the embedding of *all* the data $\mathbf{X}$ we want to find the largest (or smallest) eigenvalues of the full matrix $\mathbf{G}$, i.e we want $\mathbf{U}_s$ where $\mathbf{G} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}'$ so that $\mathbf{U}_s$ are the s eigenvectors corresponding to the s largest (or smallest) eigenvalues of the full matrix $\mathbf{G}$.

The Nyström extensions tells us how to approximate $\mathbf{U}_s$ and $\mathbf{\Sigma}_s$ using only $\mathbf{\Sigma}_{\text{train},s}$, $\mathbf{U}_{\text{train},s}$ which we found during training and also $\mathbf{G}_{21}$ which we can compute from $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$. Suppose we have $\mathbf{G}$ as above:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{\text{train}} & \mathbf{G}_{21}' \\ \mathbf{G}_{21} & \mathbf{G}_{22} \end{bmatrix}$$

Then Nyström Extension is summarized in algorithm 7. In most cases the embedding is exactly the eigenvectors of some matrix $\mathbf{G}$ for example in Diffusion Maps, Laplacian Eigenmaps, HLLE, and Isomap and so the output of 7 is already the out-of-sample extension. For LLE it is more complicated we refer to [6] for details. This extension is offered in `megaman` for the embedding algorithms provided there as well as as a standalone module.

input : $\mathbf{G}_{train}$, $\mathbf{G}_{21}$, $\mathbf{U}_{\text{train},s}$ and $\mathbf{\Sigma}_{\text{train},s}$.

output: embedding $\hat{\mathbf{U}}_{\text{test}}$

Compute completed column matrix: $\mathbf{C} \leftarrow [\mathbf{G}_{\text{train}}\mathbf{G}_{21}]'$;

Estimate the eigenvectors: $\hat{\mathbf{U}}_s \leftarrow \sqrt{\frac{l}{n}}\mathbf{C}\mathbf{U}_{train,s}\mathbf{\Sigma}_{train,s}^{\dagger}$;

Return test subset: $\hat{\mathbf{U}}_{\text{test}} \leftarrow \hat{\mathbf{U}}_s[(l+1):n,:]$;

Algorithm 7: `Nyström Extension`

4.4 Discussion

In this chapter we provided an overview of several heuristics for estimating parameters used for manifold learning as well as for performing an out-of-sample extension. The primary novelty of this chapter lies in scaling these methods to large data sets as well as combining them together in a sensible and cohesive manner which will assist scientists and researchers in performing manifold learning on large data sets. The tools that we describe in this chapter are all available in the open source software package `megaman` which we describe in chapter 5. For estimation of the bandwidth parameter we described the method of [43] which uses the Riemannian Metric estimated on the tangent plane in order to define a notion

of self-consistent bandwidth parameter in that the trivial isometry to the tangent plane ought to have low distortion. We describe the challenges with this method and proposed an algorithm 5 which allows this method to scale to large data sets. For the intrinsic dimension estimation we described a method which estimates the doubling dimension of a manifold and then we exploit the relationship between the doubling dimension and the intrinsic dimension. We demonstrate both of these methods with the galaxy spectra data set where we showed that by first estimating the radius we could use the existing distance matrix to assist in estimating the appropriate dimension for embedding these data showing good consistency between radius and dimension estimation methods. Finally we conclude the chapter with a discussion of out-of-sample extensions to manifold learning and how the Nyström extension provides a scalable solution. As mentioned, each of these three methods are provided to researchers in the open source `megaman` software.

Chapter 5

SCALABLE MANIFOLD LEARNING IN PYTHON

5.1 Manifold Learning and Big Data

Research in manifold learning is making steady progress as we discussed in the previous chapters, yet there is currently no manifold learning software library efficient enough to be used in realistic research and application of manifold learning. Since it is our goal to provide researchers with the tools to perform manifold learning on scientific data sets it is important to be able to provide them with the necessary software in order to do so. Most importantly it is essential that this software are able to scale to large data sets.

As discussed in chapter 3 Manifold learning is interested in taking a data set $\mathbf{X} \in \mathbb{R}^{n \times D}$ and embedding it, via non-linear transformation ϕ, into $\mathbb{R}^{n \times s}$. In most applications of manifold learning the observed dimension D is large, otherwise dimension reduction is unlikely to be necessary. There are two potential issues in manifold learning with respect to scaling. First it was shown in [48] that the convergence rate of $\boldsymbol{\mathcal{L}}$ from (2.4) to the Laplace Bletrami Operator $\Delta_{\mathcal{M}}$ of (2.4) is:

$$\mathcal{O}\left(\frac{1}{N^{1/2}(h^2)^{1/2+d/4}}, h^2\right)$$ [1]

In particular they uses this to say that for a constant $C(\mathcal{M})$ which depends on the manifold that the optimal bandwidth is given by:

$$h^2 = c(\mathcal{M})N^{-\frac{1}{3+d/2}}$$

which in turn implies that for a given bandwidth h the optimal sample size is:

$$N = c(\mathcal{M})h^{-(d+6)}$$

This indicates that specifically for Manifold learning the convergence rates are exponential in the intrinsic dimension d and so we require an increasingly large sample size as the dimension increases.

---

[1] where $\mathcal{O}\left(\frac{1}{N^{1/2}(h^2)^{1/2+d/4}}, h^2\right)$ indicates that there are constances C_1 and C_2 such that $|\mathcal{O}\left(\frac{1}{N^{1/2}(h^2)^{1/2+d/4}}, h^2\right)| \leq C_1 \frac{1}{N^{1/2}(h^2)^{1/2+d/4}} + C_2 h^2$ for N large and h^2 small

A second motivation for manifold learning software (and algorithms) which can scale to big data is the curse of dimensionality. As discussed above and in chapter 4 manifold learning algorithms rely on generating a neighborhood around each point containing points that are used to estimate the tangent space. It is essential, therefore, that we can sample from the manifold such that these neighborhoods are non-empty but also represent a good approximation to the tangent space (i.e. we cannot simply include all of the points). If we consider sampling points uniformly with equal spacing of 10^{-2} then in dimension $D = 1$ we can sample the unit interval $[0, 1]$ in such a manner with 10^2 points. For an arbitrary unit hyper-cube of dimension D to sample with a spacing of 10^-2 we require $(10^2)^D$ points, that is *exponential in the dimension*. Therefore if we wished to ensure that there were enough data points in boxes of this size we would need an exponential number of observations in the sampled dimension. To put this number int perspective if $D = 10$ then each coordinate can be expressed (to reasonable accuracy) with 1 byte so that a point in $[1, 0]^{10}$ is 10 bytes, we require $10^{2D} = 10^{20}$ of these points therefore we require 10^{21} bytes or 1 Billion Terabytes of data, for $D = 11$ we would require at least 100 Billion TB of data.

Combining these two implies that we generally require large data sets in order to effectively perform manifold learning so that we can decrease the bandwidth h and increase the sample size N at a sufficient rate for the convergence in [48]. It is for this reason that manifold learning algorithms must scale to large data sets but also that software for manifold learning consider the computational and memory efficiency of their implementation. Finally, as discussed in chapter 1 the size of data sets that are being gathered by modern researchers in fields such as Astronomy, Chemistry, Biology etc., are increasing in both dimension and sample size requiring more robust software in order to analyze them.

The first comprehensive attempt to bring several manifold learning algorithms under the same umbrella is the `mani` Matlab package authored by [58]. This package was instrumental in illustrating the behavior of the existing ML algorithms of a variety of synthetic toy data

sets. More recently, a new Matlab toolbox `drtoolbox`[2] was released, that implements over thirty dimension reduction methods, and works well with sample sizes in the thousands.

Perhaps the best known open implementation of common manifold learning algorithms is the `manifold` sub-module of the Python package `scikit-learn`[3] [40]. This software benefits from the integration with `scikit-learn`, meets its standards and philosophy, comes with excellent documentation and examples, and is written in a widely supported open language.

The `scikit-learn` package strives primarily for usability rather than scalability. While the package does feature some algorithms designed with scalability in mind, the manifold methods are not among them. For example, in `scikit-learn` it is difficult for different methods to share intermediate results, such as eigenvector computations, which can lead to inefficiency in data exploration. The `scikit-learn` manifold methods cannot handle out-of-core data, which leads to difficulties in scaling. Moreover, though `scikit-learn` accepts sparse inputs and uses sparse data structures internally, the current implementation does not always fully exploit the data sparsity.

5.2 A Computational Overview of Manifold Learning

This section provides a brief description of non-linear dimension reduction via manifold learning, outlining the tasks performed by a generic manifold learning algorithm, in particular with computation in mind. As in previous chapters, we assume that a set of N vectors $x_1, \ldots x_N$ in D dimensions is given (for instance, for the data in Figure 5.4, $N = 3 \times 10^6$, $D = 300$). It is assumed that these data are sampled from (or approximately from) a lower dimensional space (i.e. a *manifold*) of *intrinsic dimension* d. The goal of manifold learning is to map the vectors $x_1, \ldots x_N$ to s-dimensional vectors $y_1, \ldots y_N$, where y_i is called the *embedding* of x_i and $s \ll D$, $s \geq d$ is called the *embedding dimension*. The mapping should preserve the neighborhood relations of the original data points, and as much as possible of their local geometric relations.

---

[2] `https://lvdmaaten.github.io/drtoolbox/`

[3] `http://scikit-learn.org/stable/modules/manifold.html`

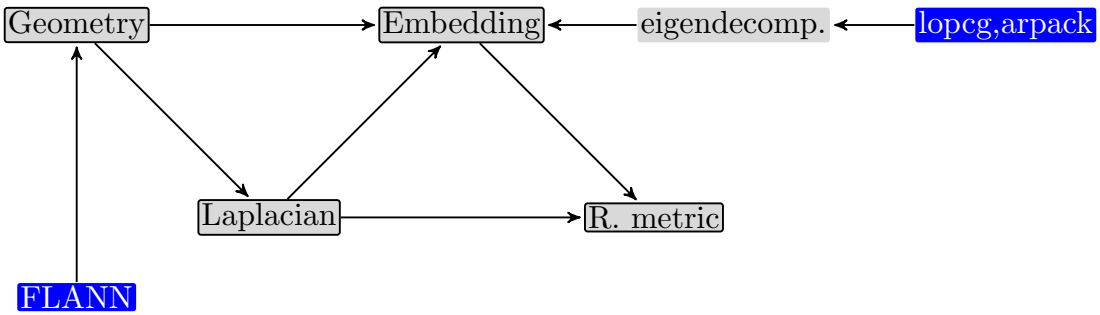

Figure 5.1: `megaman` classes (gray, framed), packages (gray, no frame) and external packages (blue). The class structure reflects the relationships between ML tasks.

From the point of view of a manifold learning algorithm (also called an *embedding algorithm*), non-linear dimension reduction subsumes the following stages.

Constructing a *neighborhood graph* G, that connects all point pairs x_i, x_j which are "neighbors". The graph G is sparse when data can be represented in low dimensions. The various methods of constructing these graphs was discussed in §2.3. Constructing the neighborhood graph is common to most existing manifold learning algorithms.

Often as another stage is to compute another matrix from the neighborhood graph/ For example, the Laplacian is directly used for embedding in methods such as the Spectral Embedding (also known as Laplacian Eigenmaps) of and Diffusion Maps of. The Laplacian is also used after embedding, in the post-processing stage. Because these operations are common to many manifold learning algorithms and capture the geometry of the high-dimensional data in the matrices S or L, we will generically call the software modules that implement them *Geometry*.

The next stage, the *Embedding* proper, can be performed by different *embedding algorithms*. There are a variety of methods some of which are discussed in §2.3.1. In general, embedding typically involves computing s eigenvectors of some $N \times N$ this matrix usually (but not always) has the same sparsity pattern as the neighborhood graph G.

Finally, for a given embedding $y_1, \ldots y_n$, one may wish to estimate the distortion incurred

with respect the original data. This can be done via the method of [41] as discussed in chapter 2 using equation (2.7), by calculating for each point y_i a *pushforward cometric* H_i. H_i is an $s \times s$ symmetric, positive definite matrix. Practically, the value $u'h_iu$, with u a unit vector, is the "stretch" at y_i in direction u of the embedding $\{y_1, \ldots, y_N\}$ with respect to the original data $\{x_1, \ldots x_N\}$. In particular, for an embedding with no stretch, that is for an *isometric* embedding, H_i should be equal to the unit matrix when the embedding dimension s and the intrinsic dimension d are the same, as discussed in chapter 3.

Even though only Riemannian Relaxation, requires the Riemannian Metric to be calculated since, the metric given by $H_1, \ldots H_N$ can be used to calculate distances, areas and volumes using the embedded data, which approximate well their respective values on the original high-dimensional data, we include an implementation of it in `megaman`.

The logical structure of Manifold learning is shown in Figure 5.1, along with some of the classes and software packages used to implement them in `megaman`.

5.3 Computational challenges

The manifold learning pipeline described in chapter 2 and in Figure 5.1 comprises two significant computational bottlenecks.

Computing the neighborhood graph, as discussed in chapter 2 involves finding the K-nearest neighbors, or the r-nearest neighbors (i.e all neighbors within radius r) in D dimensions, for N data points. This leads to sparse graph $\mathcal{G}$ and sparse $N \times N$ matrix $\mathbf{W}$ with $\mathcal{O}(d)$ neighbors per node, respectively entries per row. All $N \times N$ matrices in subsequent steps will have the same sparsity pattern as $\mathbf{W}$. Naively implemented, this computation requires $\mathcal{O}(N^2(D + \log N))$ operations.

Eigendecomposition A large number of embedding algorithms (Laplacian Eigenmaps, Diffusion Maps, Locally Linear Embedding, HLLE, Local Tangent Space Alignment, Isomap, MDS etc.) involve computing $\mathcal{O}(s)$ principal eigenvectors of an $N \times N$ symmetric, semi-positive definite matrix. This computation scales like $\mathcal{O}(N^2 s)$ for dense matrices. In addition, if the working matrices were stored in dense form, the memory requirements would scale like

N^2, and elementary linear algebra operations like matrix-vector multiplications would scale likewise.

5.4 *The* `megaman` *Package*

In this section we present and discuss a software solution, `megaman`, a new Python package for scalable manifold learning. This package is designed for performance, while inheriting the functionality of `scikit-learn`'s well-designed API [9]. `megaman` is publicly available at: `https://github.com/mmp2/megaman`. `megaman`'s required dependencies are `numpy`, `scipy`, and `scikit-learn`, but for optimal performance FLANN, `cython`, `pyamg` and the C compiler `gcc` are also required. For unit tests and integration `megaman` depends on `nose`. The most recent `megaman` release can be installed along with its dependencies using the cross-platform `conda` package manager.

5.4.1 Logical structure and classes overview

`embeddings` The manifold learning algorithms are implemented in their own classes inheriting from a base class. Included are `SpectralEmbedding`, which implements Laplacian Eigenmaps [4] and Diffusion Maps [37], Local Tangent Space Alignment [61], Locally Linear Embedding [46], and Isomap [7]. Geometric operations common to many or all embedding algorithms (such as computing distances, Laplacians) are implemented by the `Geometry` class. A `Geometry` object is passed or created inside every embedding class. In particular, the class `RiemannianMetric` produces the estimated Riemannian metric via the method of chapter 2. `eigendecomposition` (module) provides a unified (function) interface to the different eigendecomposition methods provided in `scipy`.

The `megaman` package is designed to be simple to use with an interface that is similar to the popular `scikit-learn` python package. We choose to mimic the `scikit-learn` package for several reasons. It is a popular package and interface that is well known in the machine learning community. It has a well thought out and consistent software design. It allows for easy extensions while maintaining consistency. Finally, it may be possible in the future to

merge with `scikit-learn`.

5.4.2 `megaman` *API Designed for Performance*

Most importantly, `megaman` is designed with experimental and exploratory research in mind where there must be a great deal of exploratory analysis, repeated analysis with different parameters, etc. It is well-known that manifold learning methods require large data sets to perform well, as discussed in the introduction. Unfortunately, many of these methods appear computationally unfeasible with large data sets. `megaman` attempts to overcome these technical challenges by harnessing existing tools for solving the intermediate problems and unifying them under a single framework. The FLANN C++ [36] package is used (via cython) for the pairwise neighbors calculation. Additionally sparse representations are used to minimize storage requirements and increase computing various matrices. The `megaman` package also stores intermediate steps such as the data index created by FLANN for querying nearest neighbors (in order to apply this to new data).

This allows researchers to easily repeat computations with new parameters without increasing computation time unnecessarily. Finally by manipulating the Laplacian and other embedding matrices we convert them to SSPD in order to take advantage of the matrix-free LOBPCG[25] procedure to calculate the eigendecomposition. With these, `megaman` runs in reasonable time on data sets in the millions. The design also supports well the exploratory type of work in which a user experiments with different parameters (e.g. radius or bandwidth parameters) as well as different embedding procedures; `megaman` caches any re-usable information in both the `Geometry` and `embedding` classes.

We make a quick note that in many cases the object of the eigendecomposition is a matrix of the form: $\mathbf{P} = \mathbf{D}^{-1}\mathbf{S}$ where $\mathbf{D}$ is a diagonal "degree" matrix and $\mathbf{S}$ is a symmetric similarity matrix (usually either the Guassian affinity matrix or some symmetrically normalized version of this). This matrix $\mathbf{P}$ is not symmetric positive semi-definite and so as given cannot be solved using a fast SSPD solver like LOBPCG. We use the following Proposition from [24]:

Proposition 22. *(Meilă [23] (Chapter 7)) Let* $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{S}\boldsymbol{D}^{-1/2}$*, let the eigenvalues of* $\boldsymbol{P}$ *be* $1 = \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq -1$ *and let the corresponding eigenvectors be* $v_1, \ldots v_n$ *similarly let the eigenvalues of* $\boldsymbol{L}$ *be* $\mu_1 \leq \mu_2 \leq \ldots \mu_n$ *and* $u_1, \ldots, u_n$ *be the corresponding eigenvectors. Then we have*

$$\lambda_i = 1 - \mu_i \qquad \textit{and} \qquad v_i = \boldsymbol{D}^{-1/2} u_i$$

Therefore since $\mathbf{L}$ is symmetric positive definite (positive definiteness follows from the proposition) we can use a SSPD solver to find the eigenvectors of $\mathbf{L}$ and then apply the proposition to find the eigenvectors of $\mathbf{P}$. This allows us to use the much faster and more stable SSPD solvers in all cases – particularly in Laplacian Eigenmaps and Diffusion Maps for a wide variety of different (potentially asymmetric) Laplacian matrices.

We made several design and implementation choices in support of scalability. We summarize these design features below:

- Sparse representation are used as default for optimal storage and computations.

- We incorporated state of the art Fast Library Approximate Nearest Neighbor search algorithm by [36] (can handle Billions of data points) with a cython interface to Python allowing for rapid neighbors computation.

- Intermediate data structures (such as data set index, distances, affinity matrices, Laplacian) are cached allowing for testing alternate parameters and methods without redundant computations.

- By converting matrices to sparse symmetric positive definite (SSPD) in all cases, `megaman` takes advantage of `pyamg` and of the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) package as a matrix-free method [25] for solving generalized eigenvalue problem for SSPD matrices. Converting to symmetric matrices also improves the numerical stability of the algorithms.

- Lazy evaluation in post-processing (specifically the `rmetric`/"stretch" evaluation)

5.4.3 Designed for extensions

`megaman`'s interface is similar to that of the `scikit-learn` package, in order to facilitation easy transition for the users of `scikit-learn`.

`megaman` is object-oriented and modular. For example, the `Geometry` class provides user access to geometric computational procedures (e.g. fast approximate radius neighbors, sparse affinity and Laplacian construction) as an independent module, whether the user intends to use `embedding` methods or not. `megaman` also offers the unified interface `eigendecomposition` to a handful of different eigendecomposition procedures (dense, `arpack`, `lobpcg`, `amg`). Consequently, the `megaman` package can be used for access to these (fast) tools without using the other classes. For example, `megaman` methods can be used to perform the Laplacian computation and embedding steps of spectral clustering. Finally, `Geometry` accepts input in a variety of forms, from data cloud to similarity matrix, allowing a user to optionally input a pre-computed similarity.

This design facilitates easy extension of `megaman`'s functionality. In particular, new embedding algorithms and new methods for distance computation can be added seamlessly. More ambitious possible extensions are: neighborhood size estimation, dimension estimation, Gaussian process regression.

Finally, the `megaman` package also has a comprehensive documentation with examples and unit tests that can be run with nosetests to ensure validity. To allow future extensions `megaman` also uses Travis Continuous Integration[4].

5.4.4 Additional Features

In addition to the manifold learning algorithms described above, `megaman` includes many other features. Some of the features are designed to assist researchers in performing manifold learning, others are related procedures that benefit from the tools offered by `megaman`.

For helping researchers perform manifold learning `megaman` offers the following additional

[4]https://travis-ci.org/

features.

- **Radius estimation**: the bandwidth parameter determines the size of the neighborhood graph. It's an essential parameter that determines how close points must be to be considered "neighbors". `megaman` offers a tool based on geometric self-consistency for estimating an optimal radius. The tool also allows for parallel processing in order to further speed up the estimation. As described in chapter 4.

- **Dimension (intrinisic and embedding) estimation**: the embedding dimension s and or the intrinsic dimension d are required parameters for most manifold learning algorithms although they are, in most cases, unknown. Based on relationships with the "doubling-dimension" `megaman` offers a tool for estimating the intrinsic dimension of the manifold.

- **Nyström Method**: There are two uses for the Nyström Extension in manifold learning. The first is for estimating a dense eigendecomposition based on a subset. This can be useful for embedding algorithms such as Isomap which don't naturally scale. Second, in some applications of manifold learning the goal is to pre-process the data for a prediction algorithm. The Nyström method offers a natural estimate which can be used to do apply the mapping learned on training data to a new test data set.

See chapter 4 for details about the methods described above including some synthetic data examples and real examples using the astronomy data discussed below.

Although not manifold learning there are a family of clustering algorithms, called spectral clustering, which use many of the features provided by `megaman` , the neighborhood graph, affinity matrix, Laplacian matrix, and eigendecomposition. Because manifold learning and spectral clustering tasks are so closely related – in fact the penultimate step of Spectral Clustering is to perform an embedding – it makes sense to provide an interface to them in `megaman`. In particular `megaman` offers the following methods:

- **Spectral Clustering**: Although not otherwise related to manifold learning in goal, Spectral clustering requires the use of a Laplacian matrix based on an Affinity matrix as well as computing the eigendecomposition – thereby computing an embedding – of this matrix. These two steps are implemented in `megaman` . Along with an independent implementation of the k-means clustering algorithm [30] which takes advantage of the fast orthogonal intialization [38], `megaman` offers an interface to the Spectral Clustering algorithm as described in [24]

- **Directed Spectral Clustering**: Sometimes the graph that the researchers are interested in clustering is a directed graph. Examples include citation networks and "following" graphs on e.g. Twitter. In addition to normal Spectral Clustering, `megaman` offers an interface to a the Directed Spectral Clustering described in [32].

See chapter 6 for details about the spectral clustering methods and their relationship to Manifold Learning as well as an application using a citation network from Jstor.

5.5 Benchmarking

In this section we compare the performance of the `megaman` package as the dimension D and the sample size N increase. The one other popular comparable implementation of manifold learning algorithms is the `scikit-learn` package. To make the comparison as fair as possible, we choose Laplacian Eigenmaps, which is empirically the fastest of the usual ML methods, method for the comparison, with radius-based neighborhoods and the Locally-Optimized Block-Preconditioned Conjugate Gradient (LOBPCG) eigensolver. Note that with the default settings, `scikit-learn` would perform slower than in our experiments.

We display total embedding time (including time to compute the graph $\mathcal{G}$, the Laplacian matrix $\boldsymbol{\mathcal{L}}$ and the embedding $\mathbf{Y}$[5]) for `megaman` versus `scikit-learn`, as the number of samples N varies (Figure 5.2) or the data dimension D varies (Figure 5.3). All benchmark

[5]For `megaman` we also compute the Riemannian metric estimate at each point; this time is negligible compared to the total time to obtain the embedding.

computations were performed on a single desktop computer running Linux with 24.68GB RAM and a Quad-Core 3.07GHz Intel Xeon CPU. We use a relatively weak machine to demonstrate that our package can be reasonably used without high performance hardware.

We display total embedding time (including time to compute the graph $\mathcal{G}$, the Laplacian matrix and the embedding) for `megaman` versus `scikit-learn`, as the number of samples N varies (Figure 5.2) or the data dimension D varies (Figure 5.3).

The experiments show that `megaman` scales considerably better than `scikit-learn`, even in the most favorable conditions for the latter; the memory footprint of `megaman` is smaller, even when `scikit-learn` uses sparse matrices internally. The advantages grow as the data size grows, whether it is with respect to D or to N.

The two earlier identified bottlenecks, distance computation and eigendecomposition, dominate the compute time. By comparison, the other steps of the pipe-line, such as Laplacian and Riemannian metric computation are negligible. The distance calculation scales with both sample size N and dimension D of the observed data where scaling is comparatively worse for D than for N. The eigendecomposition, however, depends only on N as it operates on an $N \times N$ matrix, however it generally scales worse with N than the distance computation.

The primary bottlenecks to the embedding procedures are the neighborhood graph construction and the eigendecomposition. In order to optimize the speed of the package on large data use option 'distance_method' = 'cyflann' should be chosen which directly interfaces with the FLANN C++ package to compute the neighborhood graph. The fastest method for computing the eigendecomposition is with LOBPCG using preconditioning from the pyamg package, option 'eigen_decomp' = 'pyamg'. Finally, not all embedding method scale equally. The fastest method for use on largest scale data is the Laplacian Eigenmaps/Diffusion Maps. Locally Tangent Space alignment scales better than LLE and Isomap itself is fundamentally non-scalable as it requires the eigendecomposition of an *dense* $N \times N$ matrix of graph shortest paths.

We also report run times on two real world data sets. The first is the `word2vec` data

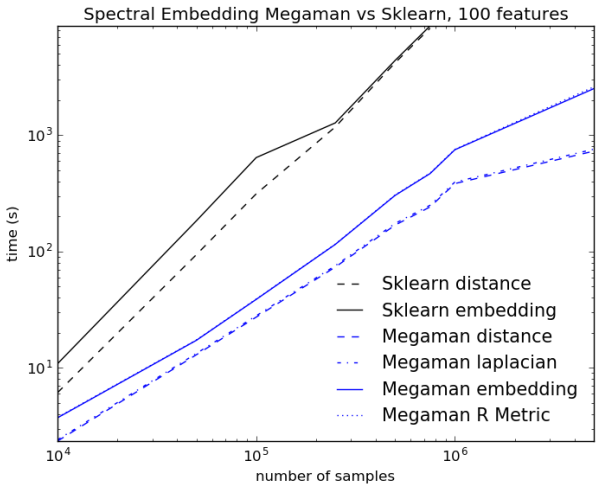

Figure 5.2: **Run time vs. data set size** N for fixed $D = 100$ and $h = \frac{5}{N^{1/8}} + D^{1/4} - 2$. The data is from a Swiss Roll (in 3 dimensions) with an additional 97 noise dimensions, embedded into $s = 2$ dimensions by the `SpectralEmbedding` algorithm. By $N = 1,000,000$ `scikit-learn` was unable to compute an embedding due to insufficient memory. All `megaman` run times (including time between distance and embedding) are faster than `scikit-learn`.

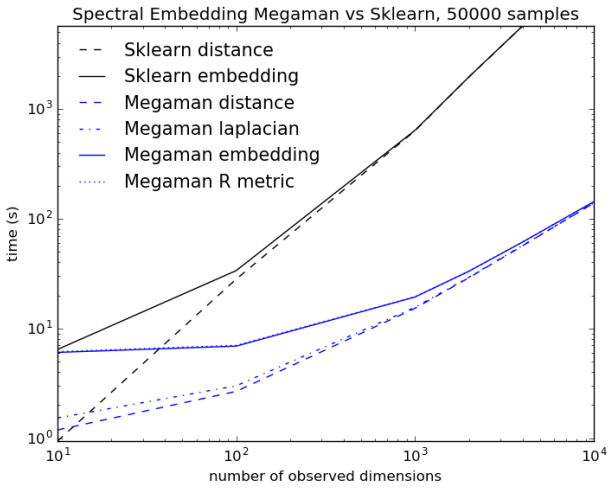

Figure 5.3: **Run time vs. data set dimension** D for fixed $N = 50,000$ and $h = \frac{5}{N^{1/8}} + D^{1/4} - 2$. The data is from a Swiss Roll (in 3 dimensions) with additional noise dimensions, embedded into $s = 2$ dimensions by the `SpectralEmbedding` algorithm. By $D = 10,000$ `scikit-learn` was unable to compute an embedding due to insufficient memory. All `megaman` run times (including time between distance and embedding) are faster than `scikit-learn`.

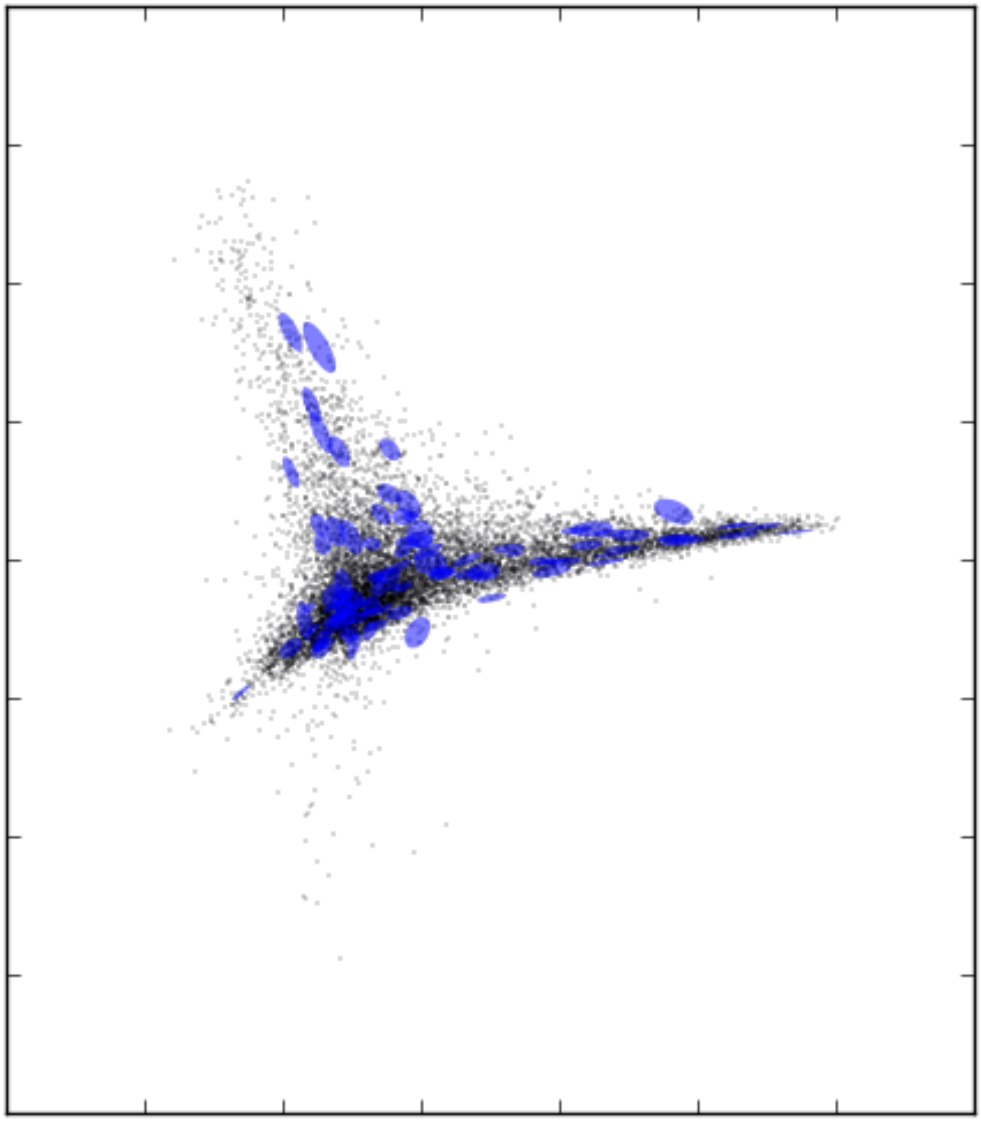

Figure 5.4: 3,000,000 words and phrases mapped by `word2vec` into 300 dimensions were embedded into 2 dimensions using Spectral Embedding. The plot shows a sample of 10,000 points displaying the overall shape of the embedding as well as the estimated "stretch" (i.e. pushforward Riemannian co-metric) at various locations in the embedding.

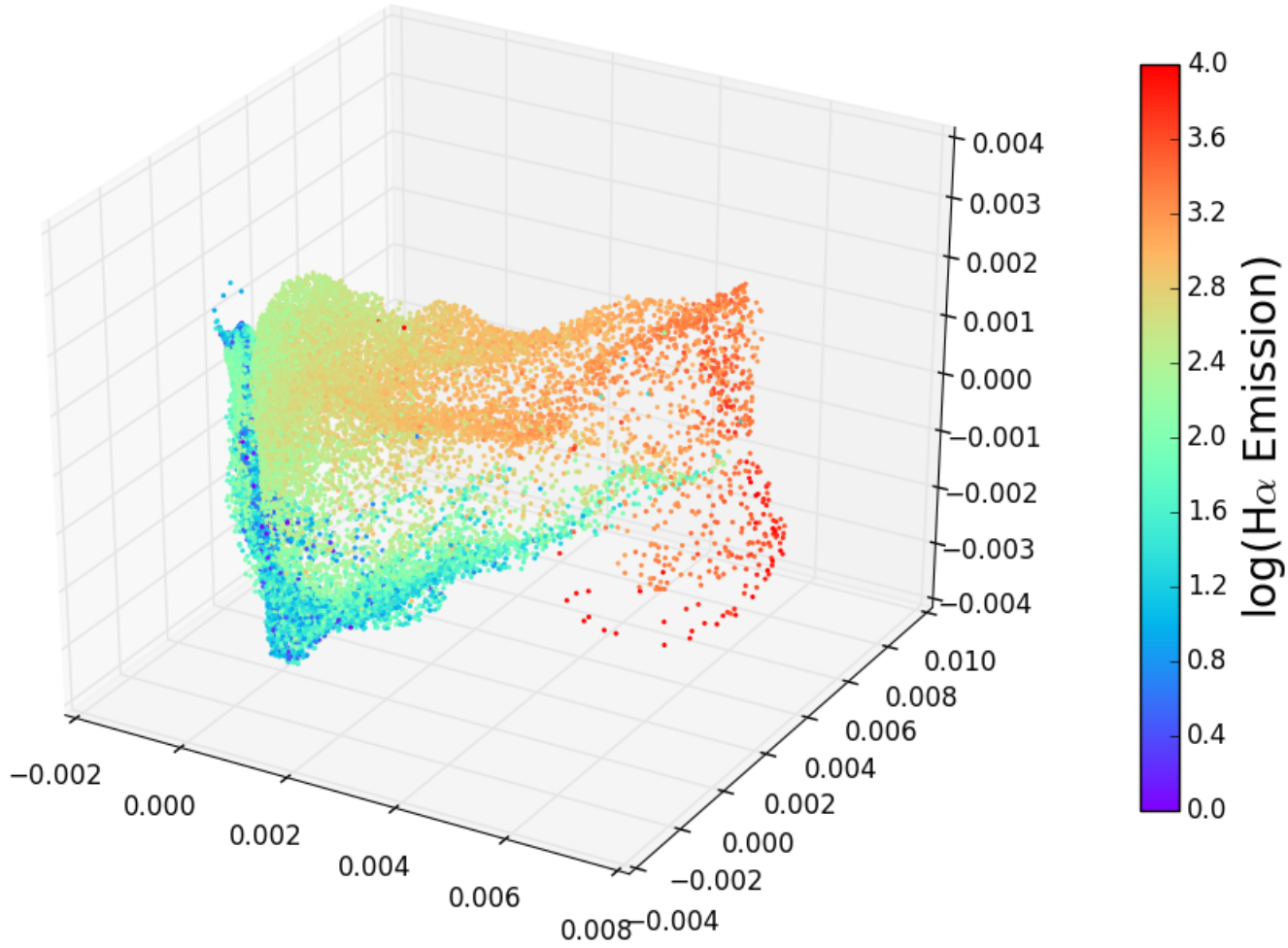

Figure 5.5: A three-dimensional embedding of the main sample of galaxy spectra from the Sloan Digital Sky Survey (approximately 675,000 spectra observed in 3750 dimensions). Colors in the above figure indicate the strength of Hydrogen alpha emission, a very nonlinear feature which requires dozens of dimensions to be captured in a linear embedding [59]. A nonlinear embedding such as the one shown here is able to capture that information much more succinctly [55] and using the entire data set. (Figure by O. Grace Telford.)

Table 5.1: Breakdown of Embedding Time by Step

			Run time [min]			
Dataset	**Size N**	**Dimensions D**	**Distances**	**Embedding**	**R. metric**	**Total**
Galaxies	0.7M	3750	190.5	8.9	0.1	199.5
Word2Vec	3M	300	107.9	44.8	0.6	153.3

set[6] which contains feature vectors in 300 dimensions for about 3 million words and phrases, extracted from Google News. The vector representation was obtained via a multilayer neural network by [34]. The second data set contains galaxy spectra from the Sloan Digital Sky Survey[7] [1], preprocessed by first moving them to a common rest-frame wavelength and filling-in missing data following [55] but using the more sophisticated weighted PCA algorithm of [16]. See chapter 1 for a longer discussion of these data. For the purpose of illustration, Figures 5.4 and 5.5 display the embeddings obtained.

5.6 Discussion

Manifold learning is data intensive; as discussed in §5.1 manifold learning depends asymptotically on the sample size N like $N^{1/(d/2+3)}$, hence requires large amounts of data when the intrinsic dimension d is larger than 1 or 2. Moreover dimension reduction is more beneficial when the input data is high-dimensional.

It was a widely accepted view in applied machine learning that manifold learning methods like the ones described above are "too computationally complex for big data". However, as we have demonstrated in this chapter and the previous chapters that this view is unfounded, or at best founded on a very naive model of computation, that does not exploit the *locality* that is at the basis of manifold learning. From the mathematical, scientific and intuitive

[6] Downloaded from `GoogleNews-vectors-negative300.bin.gz` on `https://code.google.com/archive/p/word2vec/`.

[7] `www.sdss.org`

points of view, a *manifold* is the name for a "collection of low-dimensional local patches". Our project extends and exploits this natural view of manifolds in the realm of computation and algorithms.

More specifically and pragmatically, it is evident from above that for each individual computational challenge some scalable solutions exist. Non-linear embedding is no harder than linear embedding by means of PCA. The main difference is that the eigendecomposition is done on a *sparse matrix* (with sparsity that depends on the embedding dimension d and not on the data set size, while standard PCA typically works from a dense matrix. Finding neighborhood graphs, and solving linear systems are generic problems whose numerics have been studied and optimized for decades now.

`megaman` puts in the hands of scientists and methodologists alike tools that enable them to apply state of the art manifold learning methods to data sets of realistic size. The package is easy to use for all `scikit-learn` users, it is extensible and modular. We hope that by providing this package, non-linear dimension reduction will be benefit those who most need it: the practitioners exploring large scientific data sets.

Chapter 6

SCALING SPECTRAL CLUSTERING

6.1 Spectral Clustering Background

In this section we discuss some novel methods for pre-processing and scaling the clustering algorithms, specifically working with the Spectral Clustering algorithms of [24] and the directed spectral clustering algorithm of [32]. As implied by the name, Spectral clustering algorithms involve computing an eigendecomposition given an initial "affinity" matrix $\mathbf{A}$, for example the Guassian Neighborhood Graph such as (2.3) would be an example of such an affinity matrix but in general the matrix $\mathbf{A}$ is an $n \times n$ matrix with n being the number of nodes in the graph (usually related to individuals, papers, books, etc.) and the edge weight $\mathbf{A}_{ij}$ encodes a notion of similarity between node i and node j. For example if this data came from Facebook the nodes could be individuals and $\mathbf{A}_{ij} = 1$ if individual i and j are friends. For the symmetric clustering problem discussed in [24] $\mathbf{A}$ is assumed to be symmetric, that is, the relationship from i to j is equal to the relationship from j to i. In many applications where a clustering (i.e. a partition) of the data is desired the graph is not symmetric. Examples include citation graphs (where paper i citing paper j generally does not mean that paper j cites paper i) or social community networks with a "follow" relationship like Twitter or Instagram. The spectral algorithm in [32] is aimed at solving the asymmetric clustering problem.

Before we discuss our contributions to spectral clustering we first describe the problem which spectral clustering is trying to solve. First of all given a set of nodes $V = [1, \ldots .n]$ we seek a "good" partition of these nodes into clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$. To make this more precise, the spectral clustering algorithm is motivated as a relaxation of trying to minimize the normalized cut which is defined as follows. Given an affinity matrix $\mathbf{A}$ with $\mathbf{A}_{ij} \geq 0$ and a set of non-overlapping clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ where the clusters for a partition of the nodes define the cut between two (disjoint) clusters $\mathcal{C}$ and $\mathcal{C}'$ as:

$$Cut(\mathcal{C}, \mathcal{C}') = \sum_{i \in \mathcal{C}} \sum_{j \in \mathcal{C}'} A_{ij}$$

that is, the cut is the sum of all of the weights of the edges between $\mathcal{C}$ and $\mathcal{C}'$. Given a

clustering $\mathcal{C}$ define the degree of cluster C_k to be:

$$D_k = \sum_{i \in C_k} A_{ij}$$

Then the Normalized Cut is given by:

$$NCut(\mathcal{C}) = \sum_{k=1}^{K} \frac{Cut(\mathcal{C}_k, V \setminus \mathcal{C}_k)}{D_k} = \sum_{k=1}^{K} \frac{\sum_{i \in \mathcal{C}_k} \sum_{j \notin \mathcal{C}_k} A_{ij}}{D_k} \tag{6.1}$$

$$= \sum_{k=1}^{K} \frac{\sum_{k' \neq k} (\sum_{i \in \mathcal{C}_k} \sum_{j \in \mathcal{C}_{k'}} A_{ij})}{D_k} = \sum_{k=1}^{K} \sum_{k' \neq k} \frac{Cut(\mathcal{C}_k, \mathcal{C}_{k'})}{D_k} \tag{6.2}$$

Then the goal of Spectral Clustering is to find a clustering of the data $\mathcal{C}$ which minimizes $NCut(\mathcal{C})$. Solving this problem directly by enumeration, especially for large k, is clearly intractable and so Spectral Clustering provides a lower bound of this cost function which has a solution by computing an eigendecomposition.

The Spectral Clustering algorithm as described in [24] involves computing the eigenvectors (i.e. an embedding) of the Laplacian (in this case $\mathbf{L} = \mathbf{D}^{-1}\mathbf{A}$) into $\mathbb{R}^k$ which is itself a manifold learning procedure. The spectral clustering algorithm of [24] is summarized in algorithm 8 `SymmetricSpectralClustering - Meilă and Shi`.

Input : Affinity matrix $\mathbf{A}$, number of clusters K

Output: Clustering $\mathcal{C}$ of nodes $V = \{1, \ldots, n\}$

Calculate node degrees $d_i = \sum_{j=1}^{n} A_{ij}$;

Compute Random Walk Laplacian $P_{ij} = \frac{A_{ij}}{d_i}$, $\mathbf{P} = [P_{ij}]$, for $i, j = 1, \ldots, n.$;

Compute the largest K eigenvalues $\lambda_1, \geq \cdots \geq \lambda_K$ and eigenvectors $v_1, \ldots, v_k$ of $\mathbf{P}$;

Define the embedding $\mathbf{Y} = [v_2, \ldots, v_k] \in \mathbb{R}^{n \times (K-1)}$;

Compute a clustering $\mathcal{C}$ of $\mathbf{Y}$ by the `K-Means` algorithm [30].

Algorithm 8: `SymmetricSpectralClustering - Meilă and Shi`

Additionally in this chapter we consider spectral clustering for asymmetric graphs as well. Here we follow [32] who define a slightly different *weighted cut* and a proposed spectral

solution. The weighted cut is given by:

$$WCut(\mathcal{C}) = \sum_{k=1}^{K} \sum_{k' \neq k} \frac{Cut'(\mathcal{C}_k, \mathcal{C}_{k'})}{T_k}$$

Where $Cut'(\mathcal{C}_k, \mathcal{C}_{k'}) = \sum_{i \in \mathcal{C}_k} \sum_{j \in \mathcal{C}_{k'}} T'_i A_{ij}$, T_i are input *volume* weights $T_k = \sum_{i \in \mathcal{C}_k} T_i$ and T'_i are input *row* weights. By taking $T = T' = D$ and $A = D^{-1}A$ we recover the $NCut(\mathcal{C})$ from (6.1). In the directed case, however we assume that $\mathbf{A}$ is asymmetric (for example in a citation graph). The clustering is still achieved by a spectral embedding but of a modified matrix. We summarize the algorithm in `DirectedSpectralClustering - Meilă and Pentney` 9.

Input : Affinity matrix $\mathbf{A}$, number of clusters K, volume weights T, row weights T'

Output: Clustering $\mathcal{C}$ of nodes $V = \{1, \ldots, n\}$

Re-weight the Affinity matrix: $\mathbf{A} \leftarrow T'\mathbf{A}$;

Calculate node degrees $d_i = \sum_{j=1}^{n} A_{ij}$ for $i = 1, \ldots, n$, set $\mathbf{D} \leftarrow \text{diag}(d_i)$ for $i = 1, \ldots n$;

Compute Hermitian Matrix: $\mathbf{H} \leftarrow \frac{1}{2} T^{-1/2}(2\mathbf{D} - \mathbf{A} - \mathbf{A'})T^{-1/2}$;

Compute the smallest K eigenvalues $\lambda_1, \geq \cdots \geq \lambda_K$ and eigenvectors $v_1, \ldots, v_k$ of $\mathbf{H}$;

Define the embedding $\mathbf{Y} = [v_2, \ldots, v_k] \in \mathbb{R}^{n \times (K-1)}$;

Compute a clustering $\mathcal{C}$ of $T^{-1/2}\mathbf{Y}$ by the `K-Means` algorithm [30].

Algorithm 9: `DirectedSpectralClustering - Meilă and Pentney`

Spectral Clustering and Manifold Learning share many similarities: They both work with a distance/affinity matrix, they both construct a Laplacian matrix, in many cases they involve an eigendecomposition. In fact, as mentioned, one of the steps of Spectral Clustering is actually to perform an embedding and then cluster the resulting embedding. This, therefore, ties into manifold learning and in particular can be accelerated in similar ways to how manifold learning was accelerated in `megaman`, i.e. by considering the affinity and eigendecomposition steps. The goals of manifold learning and spectral clustering are different. Manifold Learning wants to find a single embedding of the data and therefore performs best when the data lie in a single connected component whereas Spectral Clustering

is trying to tease apart separate components/clusters and a task which is easier with a disconnected graph that is broken apart into clusters. Thus, even though the algorithms are similar they are appropriate for different types of data distributions.

6.2 Challenges of Spectral Clustering

In this chapter we address two challenges for spectral clustering. In particular we consider dealing with outliers and bridges. In the case of spectral clustering we consider outliers to be nodes which are isolated or have very small degree. When performing the eigendecomposition of the Laplacian (or other matrix) when the graph contains nodes of this variety can create computation challenges, in particular these outliers will dominate the embedding so that the clustering effectively becomes outlier or not-outlier. While it is possible to run a first spectral clustering to attempt to remove these points and then re-run the algorithm this is not ideal for two reasons. First it will require performing two, potentially expensive, eigen-decompositions of a (quite possibly large) matrix.

Another challenge with spectral clustering on large data sets is dealing with bridges. Before defining a bridge mathematically we give the following intuition. It is common in most observed networks (like friendship networks or citation graphs) that the data, unadjusted, tends to fall into one large connected component. This phenomenon is fairly simple to explain: suppose that there are two islands A and B. Everyone on island A is only connected to individuals on island A and similarly for island B. This network therefore has two connected components, A and B. If a single person on island A, however, creates a connection to a single person on island B then two distinct connected components merge to a single connected component. In practice this means that, for example when dealing with a friendship network from Facebook, it takes only one person having a friend in another country to merge components. Or it takes only one paper citing another outside of its field to connect the two fields.

That is a bridge is a an *edge* e in a graph such that if the the edge e is removed the number of connected components in the graph increases by one. In the previous example

there is one large connected component but if the edge from the individual in island A to the individual in island B is removed then there will be two connected components. Since the goal of clustering is separating a graph into connected components using an efficient method to remove these bridges before clustering can significantly reduce computation time. Additionally, if all bridges can be removed then the graph will fall into separate connected components simplifying the task of clustering.

A more challenging problem than identifying bridges is identifying what we will call $s-$bridges. Similar to a bridge an s-bridge will be a *collection* of s edges such that removal of all s edges increases the number of connected components by 1. A bridge is therefore just a 1-bridge in this terminology. An example in our Island analogy is that instead of one person having a friend on both islands we have s edges (where s is small) which connect people from both islands. If we remove all s edges we have two connected components again. To re-iterate since the goal of a clustering algorithm is to separate the graph into connected components these bridges and "s-bridges" can be a significant challenge for the spectral clustering method to overcome. In the next section we propose a pre-processing algorithm which can operate on very large graphs, i.e. without loading the graph into memory. We demonstrate this algorithm with an example using a large, sparse, directed citation graph from Jstor[57].

6.3 Pre-processing Graphs for Spectral Clustering

The primary goal of a pre-processing step is to identify and remove bridges and outliers. In particular we consider the case when the data, in this case an (unweighted) list of edges, is large and it is impossible (or inadvisable) to load the entire data set into memory. To find bridges we consider spanning trees of the network. Any spanning tree will contain all bridges. To see this consider a spanning tree (i.e. a connected graph that contains all the nodes with no cycles). Since the tree spans all of the nodes if there is a connection between a component A and a component B then the bridge to connect them must be included in the spanning tree since this edge cannot create a cycle and is the only way to span the graph (i.e. connect two components).

If there is an s-bridge, that is a set of s edges that connect two components then the spanning tree will always contain one of these edges. Spanning trees will also trivially always contain leaf nodes – i.e. nodes of degree 1, since the spanning tree contains all nodes it must contain these edges as they are the only way to connect the node to the graph. These edges are also trivial bridges – connecting the network with the leaf node. An example leaf node would, in a citation network, a paper that cites a single paper in the network but is itself not cited by anyone else in the network.

A single spanning tree will help for removing bridges and leaf nodes since it will always contain these types of edges. Spanning trees, however, will also contain plenty non-bridge edges which we *don't* want to remove. Therefore we cannot create a single spanning tree and remove all edges that fall into that tree since many non-bridges may be removed in this process. The next idea is then to create several *random spanning trees*. Since bridges will *always* be contained in a spanning tree any edge that is a bridge will be contained in *all* random spanning trees. Therefore if we look at the edges that are in *all* of the spanning trees this set will contain all of the bridges and many fewer non-bridges.

How can we construct a random spanning tree? Since a minimal spanning trees can be constructed via the breadth first search algorithm given an initial node, it's possible, therefore, to perform breadth first search starting at a random node to give a random minimal (in terms of number of edges included) spanning tree. This has several downsides. (1) Unless the entire network is a single connected component a minimal spanning tree will not contain all of the nodes in the network and it is often the case in large sparse networks that the graph is disconnected. (2) In general the breadth first search algorithm requires examining a target node and then visiting *all* of the children of that node which haven't already been visited and adding those to the tree. The problem with this is that it requires either random access to an adjacency list or the ability to read the entire network into memory so that random access to the adjacency list is possible. Ultimately, this means it will be hard to compare two random spanning trees if they begin from nodes in separate connected components since both trees may cover different disconnected components. It also means it will be impossible

to scale up this algorithm to large data (i.e. that can't be fit into RAM) without challenging implementations of parallel-breadth-first-search [10].

6.3.1 Prune Graph For Clustering

In this section we describe Prune Graph For Clustering which is summarized in algorithm 12. The basic idea of the algorithm is to create several *random* spanning trees and removing edges in common between these spanning trees. The first algorithm, Streaming Random Spanning Trees summarized in 10, describes how to streaming through a file that contains all of the edges in the graph and create a random spanning tree. If the data is stored in an adjacency list this can be converted into a list of edges in a single pass through the data and without reading the entire list into memory. The next step is create a set of edges to skip by constructing several random spanning trees, as described in algorithm 11. Once these edges are removed we repeat this process without these edges. Overall we perform a fixed number of passes wherein we create a list of nodes to remove by constructing a fixed number of random spanning trees, as summarized in 12.

To construct the random spanning tree. We store the data in a Union-Find data structure of [19]. A Union Find data structure is a simple data structure equipped with the following operations:

- `MakeSet(node)`: this operation initializes a set containing a single, passed, node. This node becomes the 'reference node' for that set.

- `Find(node)`: this operation returns the 'reference node' for the set containing the passed node. This operation is made fast by using path-compression [19].

- `Union(node1, node2)`: this operation joins the set containing `node1` with the set containing `node2`. It does this by equating the reference nodes returned from `Find()`.

The UnionFind data structure can operate quickly (i.e. `Find` is amortized $\mathcal{O}(1)$) by using path compression and ranking the independent sets depending on depth of nodes. The

UnionFind data structure is useful because it immediately tells us if two nodes are connected (in the same connected component by testing if they have the same reference node), that is, if Find(u) = Find(v). Streaming The Random Spanning Tree algorithm in 10 is then simple and can be summarized as follows. After initializing `MakeSet` on every note so that each node is a singleton set we randomly shuffle the data file in memory and proceed to stream, that is read the data file (stored as a text list of edges) into memory line by line, through the edges in the file. When we receive a new edge u-v we check to see if adding this edge to the tree will create a cycle. To determine this we simply check if `Find(u)` = `Find(v)`. If true then u and v, are in the same connected component and so including this edge will create a cycle. If they are in different components then we include the edge in T and merge the two components. This process is repeated until we have viewed all of the edges. The tree T is guaranteed to contain all of the nodes (since we begin by including them), it is also guaranteed not to contain any cycles and so is indeed a tree. We present this algorithm in `StreamingRandomSpanningTrees` 10. Since the `Find` and `Union` operations are both amortized $\mathcal{O}(1)$ a single random spanning tree is $\mathcal{O}(|E|)$ where $|E|$ is the number of edges in the network, as the algorithm considers every edge exactly once.

As discussed above it is insufficient to use only one tree in order to identify bridges as the single tree will contain many non-bridges. Therefore we need several random spanning trees, so we repeat algorithm `StreamingRandomSpanningTrees`. The first step is to shuffle the data on disk and since the tree depends only on the order of the input edges we get a different random spanning tree every time the algorithm is run. The first of the overall algorithm 12, is to run the `StreamingRandomSpanningTrees` algorithm k times shuffling the edges each time. This results in k different random spanning trees. We then look for edge overlap in all k trees. Those edges that appear in *all* trees will most likely be bridges. These edges are then included in a set of edges to be ignored. Since we are deriving a list of "Skip Edges" we call this algorithm `RandomTreeSkipEdges` and it is summarized in algorithm 11.

---

[1] $\mathcal{O}(1)$ hash-table lookup

[2] in practice we maintain a hash table containing the edges and the number of times it is included in a

```
Input  : Data file fname consisting of edges (u, v), list of nodes nodes, hash table of
         edges to skip SkipEdges
Output: list T consisting of edges in the tree
Init   : T = ∅
Init   : Initialize UnionFind data structure
Init   : for each node n in nodes MakeSet(n)
Randomly shuffle rows of fname on disk using Unix command shuf;
Open file(fname) as f;
while file f is not empty do
    read in next edge (u, v);
    if edge not in SkipEdges[1] then
        if Find(u) ≠ Find(v) then
            T.append(edge);
            Union(u, v);
        end
    end
end
```

Algorithm 10: `StreamingRandomSpanningTree`

Not all components will be connected by a 1-bridge, as mentioned it is possible that there are a set of edges falling into an s-bridge which, together, connect two otherwise independent connected components. A random spanning tree is guaranteed to contain an edge in the s-bridge but repeated random spanning trees are not guaranteed to contain all of these edges. Therefore we repeat `RandomTreeSkipEdges` s times each time updating the list of edges to skip in `StreamingRandomSpanningTrees`, that is the next iteration streams through all the edges *except* those already in the list of edges to remove.

---

tree, edges who have count equal to k are in all trees.

Input : Data file `fname` consisting of edges (u,v), list of nodes **n**, number of random trees k, hash table of edges to skip **SkipEdges**

Output: `RemoveEdges` list of edges in data to exclude

`TreeEdges` = list();

for *tree* j *in* $1 : k$ **do**

  `TreeEdges[j]` ← `StreamingRandomSpanningTree`(`fname`, **n**, `SkipEdges`);

end

`RemoveEdges` = Find edges in all trees in `TreeEdges`[2]

Algorithm 11: `RandomTreeSkipEdges`

At the end we take the list of edges to remove as the union of all the individual skip edges, this is summarized in algorithm `PruneGraphForClustering` in 12. It is guaranteed after this process that all leaf nodes will be removed as they are trivial bridges. Additionally all non-trivial bridges are guaranteed to be removed. Additionally we hope that after this process we have removed $s-$bridges from the graph as well by sequentially reducing the size s of the set of edges that connect the two components every time we repeat the `RandomTreeSkipEdges`. The algorithm `PruneGraphForClustering` should therefore leave a more disconnected graph with outliers removed; this will then improve the performance of the clustering algorithm, any connected components created from this pruning can also be clustered separately by spectral clustering. The worst case running time of the algorithm is $\mathcal{O}(sk|e|)$ for s passes of k trees with a network of $|E|$ edges, this is worst case as it assumes that no edges are removed at each pass and so all $|E|$ edges have to be considered in the construction of every random spanning tree. Overall the algorithm is linear in the number of edges.

We note that all leaf nodes will be removed in this process, but once a clustering is obtained for the data it is a simple operation to assign a leaf node to the cluster of its unique neighbor. If this procedure results in connected components that already meet the desired criteria then the job of clustering is complete. Often though it is still required that a Spectral Embedding be performed. In the next section we discuss applying this procedure

input : Data File `fname` consisting of edges u-v, number of random trees k, number of random passes s.
output: `SkipEdges` list of edges in data to exclude
init : `SkipEdges` = list()
Stream through file once to get list of nodes **n**;
for *pass i in* $1 : p$ **do**
    `TreeOverlapEdges` ← RandomTreeSkipEdges(`fname`, **n**, k, `SkipEdges`);
    `SkipEdges`.append(`TreeOverlapEdges`)
end

Algorithm 12: `PruneGraphForClustering`

to a Citation network obtained by looking at papers stored on Jstor.

6.4 Application: Jstor Citation Network

The data set we consider in this section is the *citation network* in [57], we thank Jstor and UW DataLab[3] for providing the processed data which is the object of analysis in this section. The Jstor Corpus (http://www.jstor.org/) is a digital archive of scholarly research comprised of articles from 1545 to the present day. There are 1,787,351 unique journal papers which cite or are cited by other articles in the corpus. As this is a citation network the edges are between articles where one cites the other. This relationship is directed since paper A citing paper B does not imply paper B cites paper A, in fact it's highly unlikely that they do since most paper publish already cited papers. There are 8,227,537 directed edges. As mentioned most edges only go one direction but there are 22,564 edges that got both ways but these papers simply cite themselves and form a cycle. Based on the number of nodes this is also a *very* sparse matrix consisting of only 0.0005% of the possible edges occurring. The network tends to have more papers with out edges than papers with in-edges in that the

[3]https://datalab.ischool.uw.edu/

median number of out-edges (i.e. cited papers in the network) per paper is 2 but the median in-degree (i.e. number of citations) is only 1. It's highly skewed as well as a small number of papers compose a large amount of the citations. Additionally there are many papers in the network that are not cited by other papers in the network – i.e. have no in-degree. There are also some papers with no out-degree by construction there are no papers with both zero in and out degree. There are, however, 136 papers who cite themselves and only themselves and so make up isolated points. We summarize some of the relevant statistics in 6.1.

Table 6.1: Summary of Jstor Citation Data

Number of Papers	1,787,351
Number of citations	8,227,537
Percent non-zero values	0.0005%
Median out-degree	2.0
Median in-degree	1.0
Mean degree	4.6
Number of isolated points	136
Number of bi-directional edges	22,564

The data lies mostly in a single large connected component, 94% of the data lie in a single component. The remaining components are small connected components of size 30 or fewer. This phenomenon where very large networks appear as a single connected component was discussed in §6.2. We decided to pursue an algorithm to try and remove bridges before performing a spectral clustering. We noted that a BreadthFirst-Search takes too long to be done, also the graph is not fully connected leaving many BFS trees unsatisfactorily small and also not guaranteeing that there would be overlap between any two trees.

We used the algorithm 12 10 passes of 10 trees each the result of which is summarized table 6.2. In the first pass most of the edges that are removed are leaf nodes which are guaranteed to show up in a random spanning tree of this form by construction. There are approximately 400,000 edges that correspond to leaf nodes in these data. After the 10 passes of 10 trees we arrived at a final set of nodes consisting of 1,266,888 of the original 1,787,351. This didn't completely solve the problem as we were still left with one dominant cluster. Therefore we continued with a spectral embedding as summarized in 9. When using the

`DirectedSpectralClustering` algorithm we used $T = D$ and $T' = \mathbf{1}$.

Table 6.2: Summary of Applying `PruneGraphForClustering` to Jstor Citation Data

Pass Number	Edges Removed (leaf nodes)
1	428,891 (401,897)
2	12,546
3	10,214
4	8,824
5	7,959
6	7,287
7	6,907
8	6,400
9	5,991
10	5,592
Total Removed:	500,611
% Edges Removed:	6.08%

Our affinity matrix is not weighted since edge occurrence is noted as either 1 or 0, however the Hessian matrix in 9 is normalized giving a weighted graph, which is the object of the eigendecomposition. Even though it is the bottom eigenvectors (i.e. corresponding to the smallest eigenvalues) that are usually of interest we found that with these data both the bottom and the top eigenvectors are informative.

In particular, the most interesting clustering comes from the first and last eigenvector. We note that they both perform the obvious minimal weighted cut which is to split papers between those that have citations and those that do not have citations. Since the weight will correspond to in-degree that has been removed the papers who do not have any citations contribute 0 to this weight therefore it is not surprising to see this result. In particular $Cut(\mathcal{C}_2, \mathcal{C}_1) = 0$ making $WCut(\mathcal{C}_1, \mathcal{C}_2) = 0.26684$. If we split the data into the two clusters as determined by the bottom (and top) eigenvectors (see Figure 6.1 for a histogram of the eigenvectors and discussion on how the split was made) then the edges flowing between the two (disjoint) clusters in table 6.3.

Table 6.3: Summary of Edges between C_1 and C_2 clusters

Edges From	Edges To		# of papers
	C_1	C_2	
C_1	5,798,219	0	955,273
C_2	1,914,192	0	311,615

The top eigenvector also has a pronounced third cluster which corresponds primarily to papers which have few citations, i.e. only one or two papers cite them rather than several papers. Figure 6.1 displays the eigenvectors and how they create clusters with the data. In addition in Figure 6.3 we demonstrate that the third cluster associated with the top eigenvector is a subset of the cited cluster that contains papers which have a small number of citations (usually only one or two citations per paper) compared to the rest of the cited group which has more citations per paper.

The remaining eigenvectors all take on the form as displayed in 6.2. They appear like a double-exponential distribution with the bulk falling into the center but with notable weight in the tail. By subtracting a Gaussian to find the threshold for outliers each eigenvector produces two clusters: A and B for the outliers above and below the mean. Figure 6.2 shows these eigenvectors as well as the overlap in nodes when using different eigenvectors. For any given eigenvector A and B are disjoint. A and B contain each about 5% of the papers and on average 5% of the outgoing edges. Assuming a completely random clustering then we might expect 5% of the outbound edges from A to go to B however we observe only 2% of edges flow from A to B and vice-versa. Additionally, we note that there is an unexpectedly large overlap between A and A' clusters from two different eigenvectors. If they separated at random we would expect, again, approximately 5% of the nodes in cluster A to be in A' yet we see consistently 16% overlap between these clusters as visualized in figure 6.2.

In figures 6.3 and 6.4 we look at how the different clusters cite papers in the other. In the first figure, 6.3 we look at the edges between the clusters determined in the top eigenvector. C_2 contains no in-edges as described above. The second figure demonstrates that C_3 contains mostly papers with few citations per paper. In figure 6.4 we show the edges between the

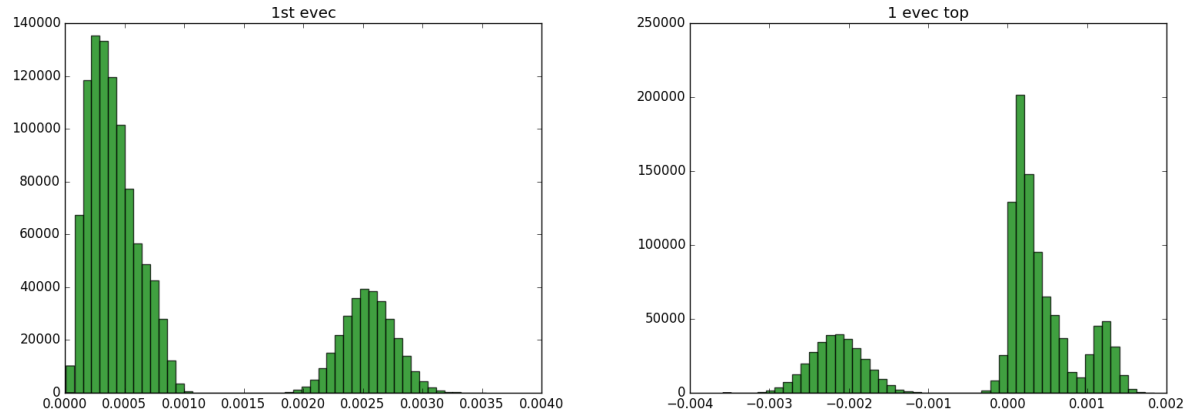

Figure 6.1: In the left hand side plot we show as histogram of the smallest eigenvector and on the right we show the histogram of the largest eigenvector. The left hand plot has notable two clusters with the larger cluster on the left hand side consisting of the papers with citations and the cluster on the right hand side consisting of papers without citations. The right hand plot is the top eigenvector and displays three clusters. The points to the left consist of not cited papers and the points to the right consist of the cited papers. Points on the far right, the small cluster, consist of papers with fewer citations in general. See the figure 6.3.

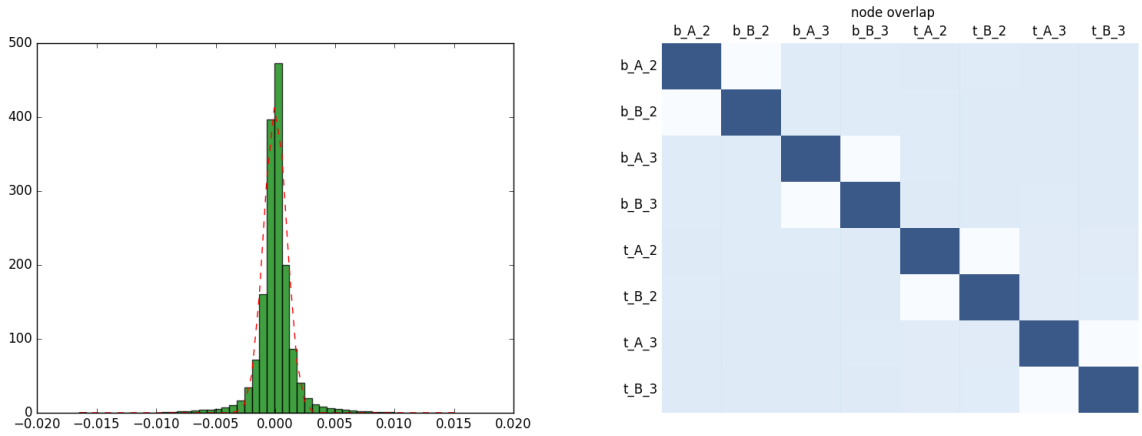

Figure 6.2: In this figure we show an example eigenvector other than the top and bottom. It is the case that the remaining eigenvectors look of this form. In the left hand plot we show how we select the A and B clusters. In particular the point at which the Guassian curve in read crosses the histogram is the threshold we set. Points below on the negative side fall into cluster A and those above the right hand side to cluster B. These approximately correspond to the 6th and 94th percentile. In the right hand plot we do this for the 2nd and 3rd largest as well as 2nd and 3rd smallest eigenvector. The naming convention is t indicates top and b indicates bottom with A, B as described above and the final numeral indicating 2nd or 3rd largest/smallest. We note that while there is overlap in the nodes between these clusters it's a much larger overlap than if you had divided the data purely at random. (16% vers 5%).

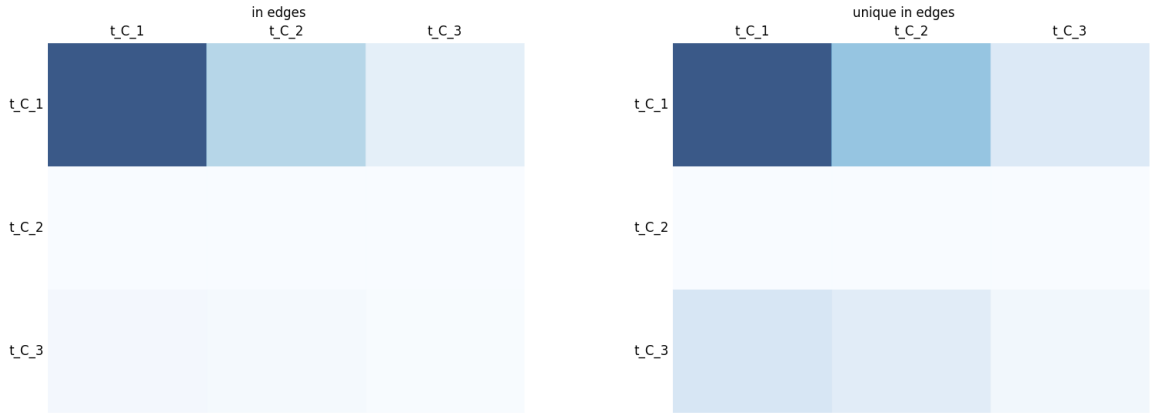

Figure 6.3: This figure demonstrates how the top eigenvector divides the data. We note that there are no incoming edges into C_2 and the vast majority of the incoming edges go into C_1 in the left hand plot. The right hand plot, however, shows *unique* in edges, that is we only count a citation to a paper once it has been cited and not any more times that other papers cite it. Here we see that C_3 gets a larger share of these unique edges. This corresponds to the majority of the papers cited in C_2 only have one or two citations per paper rather than 4 or more as is the case with the papers in C_1. So the C_1 consists primarily of highly cited papers in the network.

small A and B clusters created from the other eigenvectors. We note that there appears to be some non-uniform behavior in the out-edge heatmap indicating that the clusters created by this algorithm are not purely random.

6.4.1 Sub-Clustering Cited Papers

In the previous section we identified a set of papers C_1 which are cited (as opposed to papers which are un-cited). We were interedted in computing a sub-clustering of these papers in particular but where we can still use the information in C_2, i.e. the uncited papers. Given the asymmetric adjacency matrix A we do this by constructing the matrices $A'A$ and AA'

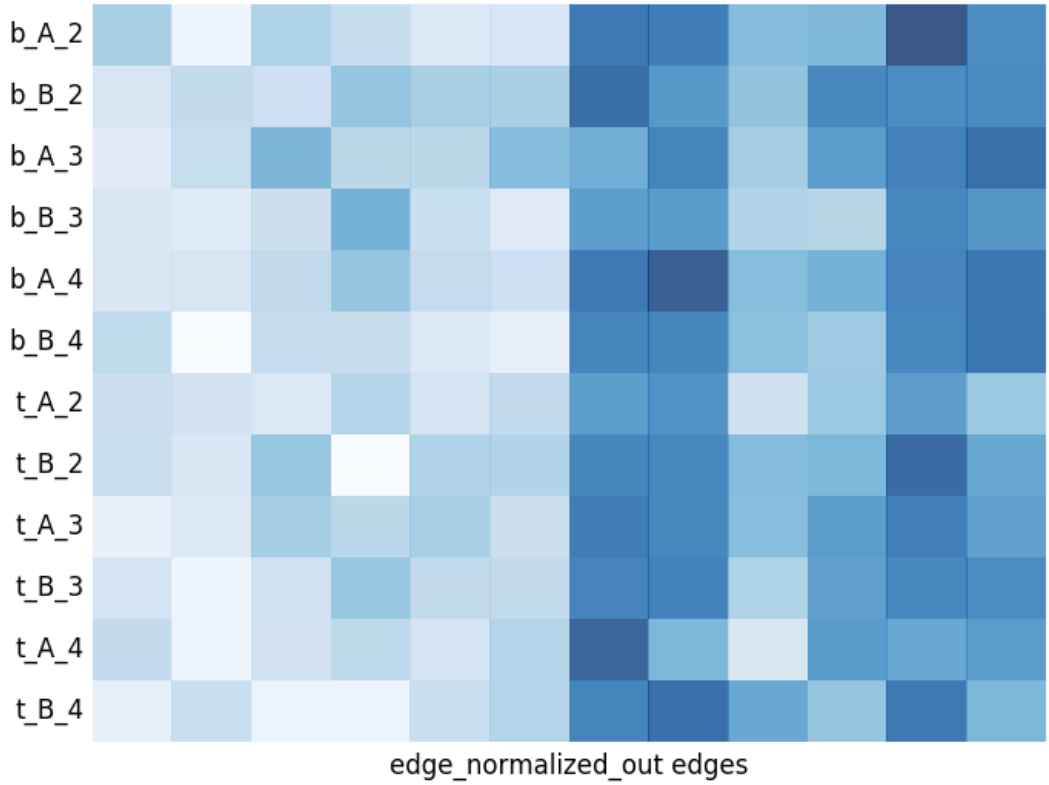

Figure 6.4: Here we look at the out-edges (in-edges is simply the transpose) of the different A and B clusters made by the other eigenvectors as discussed in figure 6.2. If the clusters were selected at random we would expect a mostly random graph. Here we see that there is a significant connection between the eigenvector clusters from the top eigenvectors, in particular they tend to have more incoming edges than those created from the bottom eigenvectors and both edges from the top clusters tend to go to clusters made from the bottom. The graph is normalized by dividing by the total number of edges in the graph. There are 4,547,567 total edges displayed in the graph out of a total 7,726,926 edges. Note that there is 16% node overlap between A and B clusters from different eigenvectors.

with all of C_1 and C_2 but we then only take rows of these matrices with respect to C_1.

Let $Q = AA'$ and let $R = A'A$ Then:

$$Q_{ij} = (AA')_{ij} = \sum_{k=1}^{n} A_{ik}(A')_{kj} = \sum_{k=1}^{n} A_{ik}A_{jk} = A_i'A_j$$

So that the ith row of Q is

$$Q_i = [A_i'A_1, \ldots, A_i'A_n]$$

And so contains information about the ith row of A and how it relates to the other rows of A. If we take Q and only consider rows and columns with respect to points $i \in C_1$ and then cluster these points we will have a clustering of points in C_1 but we will use information from C_2. Since A_i denotes in-degree then Q will be a clustering based on the in-degree of the network. Similarly if $R = A'A$ then:

$$R_{ij} = (A'A)_{ij} = \sum_{k=1}^{n}(A')_{ik}A_{kj} = \sum_{k=1}^{n} A_{ki}A_{kj} = (A^j)'A^i$$

So that the jth column of C is:

$$R^j = [(A^j)'A^1, \ldots, (A^j)'A^n]'$$

and so contains information about the jth column of A and how it relates to other columns of A. If we take C and only consider rows and columns with respect to points $i \in C_1$ and then cluster these points again we will have a clustering of points in C_1 but which will use all the information from C_2. Since A^j denotes out-degree then R will be a clustering based on the out-degree of the network.

In 6.5 we display the eigenvectors of the embedding which shows how this further splits the papers in C_1 into several smaller clusters based on the $\mathbf{A'A}$ symmetrized matrix, i.e based on out-degree. We use the first three eigenvectors to identify 8 clusters: three from the first and second eigenvectors and two from the third eigenvector (which is similar to the A/B clusters from the previous section.

In figure 6.6 we demonstrate out the out-degree relates to the clusters found in the We order the clusters by overlap. Since the size of the clusters varies, i.e. they are not all of the

same number of papers, we normalize both the overlap heatmap and the out-edge heatmap by their row-sum, i.e. number of nodes and number of out edges of each cluster respectively. There appears to be significant overlap in some of the clusters created by these eigenvectors but not always a corresponding overlap in edges such that when overlap exists it appears to be a subset of particular interest. 6.5.

6.5 Discussion

In this chapter we discussed Spectral Clustering of both directed and un-directed graphs as an approximation to minimizing weighted and multi-way cuts respectively. We focused on the challenges of spectral clustering both in general and for large graphs and emphasized the relationship between spectral clustering and manifold learning. The primary challenge of scaling spectral clustering to large data sets is similar to that of manifold learning as they both involve computing an eigendecomposition of a large sparse matrix. We were able to harness `megaman` which was designed for manifold learning to scale spectral clustering to large data sets like the Jstor citation network which includes 1.8 million papers.

More importantly, we examined the problem in spectral clustering of handling outliers and bridges in graphs which can make spectral clustering challenging. These bridges connect otherwise disconnected component such that identifying and removing them directly assists in the clustering task. We proposed a novel algorithm `PruneGraphForClustering` which aims to prune edges from a graph by removing outlier nodes and small clusters as well as bridges. We defined the notion of an s-bridge which is a set of edges (rather than a single edge) which connects two, otherwise not connected, components of a graph. The repeated nature of `PruneGraphForClustering` is aimed at identifying and removing these edges from the graph in order to improve the quality of the spectral clustering. Additionally since the task removes leaf-nodes the pre-processed graph will be of smaller size thus decreasing the computational cost of the subsequent embedding but leaving a simple way to cluster the entire graph.

We demonstrated this algorithm on the Jstor data set where we noted both the Spec-

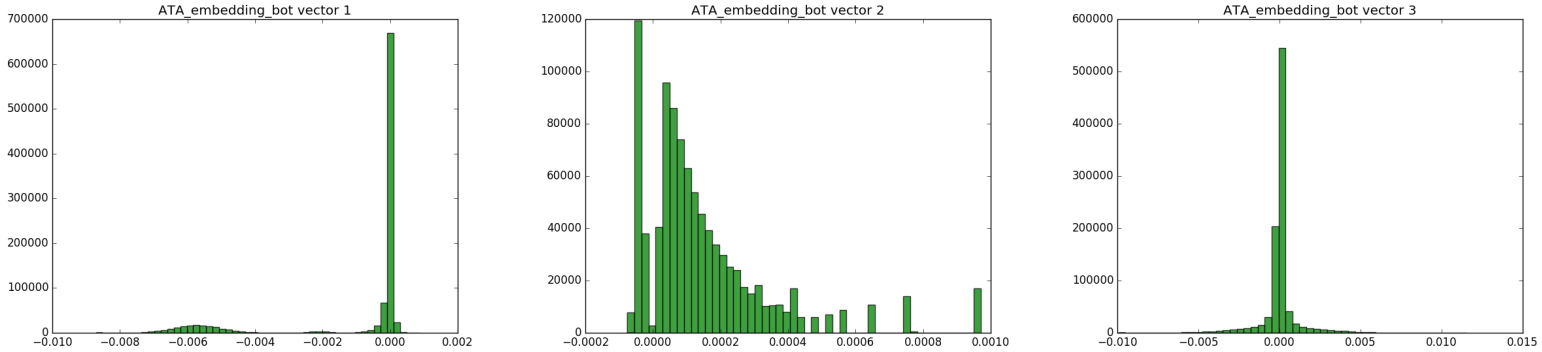

Figure 6.5: Here we consider the bottom three smallest (smallest on the left) eigenvectors from the matrix $C = \mathbf{A}'\mathbf{A}$ which values the out-degree specifically from the points in cluster C_1 from the previous section – i.e. cited papers. Here the first eigenvector indicates the presence of three clusters. The second eigenvector indicates one to the left of 0 and one or two to the right. The third eigenvector (and the remaining) appear as the far right similar to in the previous section and can be used to make A, B clusters as before.

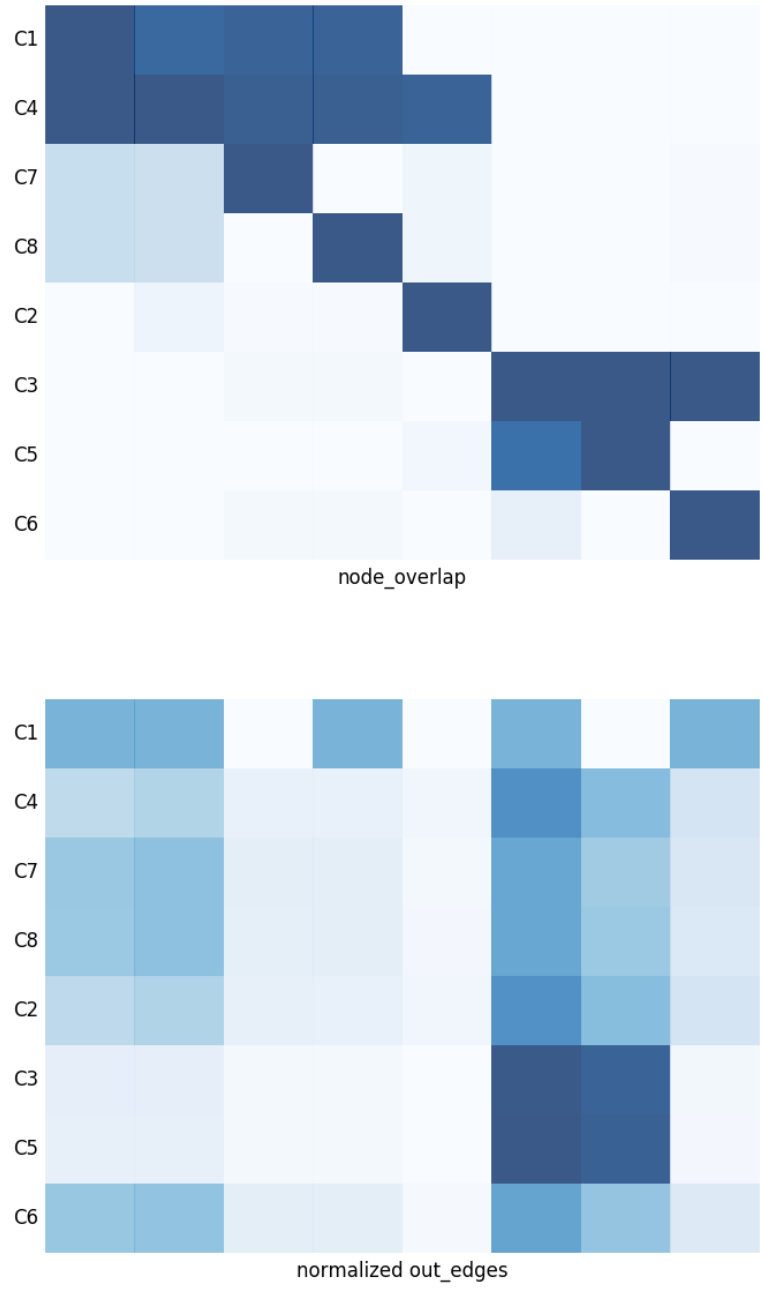

Figure 6.6: Here we show node overlap as well as the edge diagram for the clusters derived in 6.5. Since clusters are of a variety of size both plots are normalized by row. That is the rows of the top are divided by the total number of edges in that cluster and the rows on the bottom are normalized by the total out edges in that cluster. Also we note that these data consist of all the data who have citations and so all of the out edges must be distributed along the bottom plot.

traClustering algorithm cut the data by citations (not cited versus cited) but also several smaller clusters which displayed interesting and unexpected relationships. Finally we examined a symmeterization process which allowed us to cluster a subset of the graph, the cited papers identified in the directed embedding, but including information from the entire. We did this by computing in and out degree matrices $\mathbf{AA'}$ and $\mathbf{A'A}$ respectively and creating a sub-matrix containing rows and columns with respect to the subset of interest. This sub-matrix is then embedded and clustered via spectral embedding.

Chapter 7

CONCLUSION

This thesis has been concerned with the topic of *manifold learning* as well as some other related topics. Manifold learning is a complicated and vast subject with a large number of competing algorithms. As manifold learning is fundamentally an unsupervised learning problem there is not always a clear manner for distinguishing methods. Additionally, there are often several parameters which are essential for optimal performance of these manifold learning algorithms. It has been the overarching goal of this thesis to provide researchers with an understanding of the field of manifold learning as well as provide them with a guide in applying the methods of manifold learning to large scientific data sets.

To that aim it was important for us to address several things. First an adequate background and understanding of manifolds and differential geometry is required in order to cast the problem of manifold learning into the appropriate mathematical framework, second an understanding of manifold learning algorithms and isometry. Third, the ability to intelligently select parameters for manifold learning. And finally the high performance software that is required to apply manifold learning to large data sets.

In chapter 2 we described manifold learning in terms of Riemannian Manifolds. We defined what it meant to be a Riemannian Manifold and how it allowed for geometric calculations to be performed on the manifold via the Riemannian Metric. This lead to a natural definition of geometry preserving transformations called *isometries*. Finally we discuss how [42] allow us to estimate Riemannian Metrics from data.

In chapter 3 we defined one goal of manifold learning as finding an isometry ϕ from the manifold $\mathcal{M}$ to $\mathbb{R}^s$. To that end we proposed a loss function based on the estimated Riemannian Metric which encodes deviation from an isometric embedding into $\mathbb{R}^s$ We optimized this loss function via gradient descent and demonstrate its performance on both synthetic and real data. The primary innovation of this method is explicitly dealing with manifolds where the embedding dimension s is greater than the intrinsic dimension d but actively preserving the embedding to be a submanifold of $\mathbb{R}^s$ of dimension d, additionally the method appears empirically to have some noise suppressing properties.

In chapter 4 we described how scientist could estimate the parameters that most algo-

rithms in manifold learning have as input. In particular these parameters include a radius which determines how close points must be in order to be considered neighbors and the dimension of the manifold. In this chapter we described methods for estimating these parameters. For the radius we demonstrated how to scale up the geometric self-consistent radius estimation of 4 by exploiting the local nature of the Riemannian Metric and a natural parallelization scheme. For the dimension estimate we discuss a simple and computationally cheap (under the assumption that manifold learning will be performed) estimation via the doubling dimension. In both cases we demonstrate these methods using the galaxy spectra data.

In chapter 5 we presented an open source manifold learning library in python which allows scientists to apply manifold learning algorithms to large data sets. Manifold learning as a subject usually involves large data sets as the task of manifold learning is to find non-linear low dimensional representations of the data, something only necessary with high dimensional data. Due to curse of dimensionality, however, it is essential to have a sufficiently large (in terms of sample size) data sets in order to adequately perform manifold learning. In this chapter we presented an open source python package called `megaman` , which offers a solution to the challenges of scaling manifold learning to large data sets. We provide a demonstrate on several large real world data sets.

In chapter 6 we discussed a related topic to manifold learning is that of spectral clustering. In spectral clustering the goal is to cluster the data as opposed to embed the data, but the steps involved are very much related to manifold learning both via the Laplacian and computing an embedding. We described how it relates to manifold learning and so how we can exploit our understanding of scaling manifold learning to scale. The challenges of Spectral Clustering, however, are unique and in particular we examined the issue of preprocessing the graph in order to improve the quality of spectral clustering algorithms. We offer an algorithm `PruneGraphForClustering` to assist in clustering large data sets by preprocessing the data to remove nuisance edges creating bridges and outliers.

It has been the aim of this thesis to provide an overview of the manifold learning subject,

it's goals, challenges, related topics and applications. In each of these areas we have provided a description of the problem as well as offered potential solutions to each one to provide the scientists both with an understanding of the nature of manifold learning and directions for how to proceed when faced by its challenges. The topic of manifold learning is wide and largely unexplored. There exists a great deal of future work including both better methods for solving the problems described in this thesis. Topics of particular interest to scalable manifold learning include, topological analysis of manifold data and analyzing manifold data in patches.

BIBLIOGRAPHY

[1] K. N. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, S. S. Allam, C. Allende Prieto, D. An, K. S. J. Anderson, S. F. Anderson, J. Annis, N. A. Bahcall, and et al. The Seventh Data Release of the Sloan Digital Sky Survey. *Astrophysical Journal Supplement Series*, 182:543–558, June 2009.

[2] Kevork N. Abazajian et al. The Seventh Data Release of the Sloan Digital Sky Survey. *Astrophys. J. Suppl.*, 182:543–558, 2009.

[3] O. Barkan and H. Aronowitz. Diffusion maps for plda-based speaker verification. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7639–7643, May 2013.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[5] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15:1373–1396, 2002.

[6] Yoshua Bengio, Jean-FranÃğois Paiement, Pascal Vincent, Oivier Delalleau, Nicolas Le Roux, and Marie Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in Neural Information Processing Systems 16*, January 2004.

[7] Mira Bernstein, Vin de Silva, John C. Langford, and Josh Tennenbaum. Graph approximations to geodesics on embedded manifolds. `http://web.mit.edu/cocosci/isomap/BdSLT.pdf`, December 2000.

[8] I. Borg and P. Groenen. *Modern Multidimensional Scaling: theory and applications*. Springer-Verlag, 2nd edition, 2005.

[9] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

[10] Aydin Buluç and Kamesh Madduri. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 65:1–65:12, New York, NY, USA, 2011. ACM.

[11] "K. Carter, A. Hero, and R Raich". "de-biasing for intrinsic dimension estimation". *"IEEE/SP 14th Workshop on Statistical Signal Processing"*, pages 601–605, 8 2007.

[12] Lisha Chen and Andreas Buja. Local Multidimensional Scaling for nonlinear dimension reduction, graph drawing and proximity analysis. *Journal of the American Statistical Association*, 104(485):209–219, March 2009.

[13] Stefan Chmiela, Alexandre Tkatchenko, Huziel E. Sauceda, Igor Poltavsky, Kristof T. Schütt, and Klaus-Robert Müller. Machine learning of accurate energy-conserving molecular force fields. *Science Advances*, 3(5), 2017.

[14] R. R. Coifman and S. Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):6–30, 2006.

[15] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 537–546, New York, NY, USA, 2008. ACM.

[16] Ludovic Delchambre. Weighted principal component analysis: a weighted covariance eigendecomposition approach. *Monthly Notices of the Royal Astronomical Society*, 446(2):3545–3555, 2014.

[17] D. L. Donoho and C. Grimes. Hessian eigenmaps: New locally-linear embedding techniques for high-dimensional data. Technical report, Dept. of Statistics, 2003.

[18] David L. Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proc Natl Acad Sci*, 100(10):5591–5596, May 2003.

[19] Bernard A. Galler and Michael J. Fisher. An improved equivalence algorithm. *Commun. ACM*, 7(5):301–303, May 1964.

[20] Christopher Genovese, Marco Perone-Pacifico, Isabella Verdinelli, and Larry Wasserman. Minimax manifold estimation. *Journal of Machine Learning Research*, 13:1263âĹŠ–1291, May 2012.

[21] M. Hein and J.-Y. Audibert. Intrinsic dimensionality estimation of submanifolds in $\mathbb{R}^d$. In *Proceedings of the 22nd international conference on Machine learning*, ICML, pages 289–296, 2005.

[22] M. Hein, J.-Y. Audibert, and U. von Luxburg. Graph Laplacians and their Convergence on Random Neighborhood Graphs. *Journal of Machine Learning Research*, 8:1325–1368, 2007.

[23] C. Hennig, M. Meila, F. Murtagh, and R. Rocci. *Handbook of Cluster Analysis*. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. Taylor & Francis, 2015.

[24] Christian Henning, Marina Meila, Fionn Murtagh, and Roberto Rocci. *Handbook of Cluster Analysis*. Chapman and Hall/CRC, 1 edition, 2016.

[25] Andrew V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 23(2):517–541, 2001.

[26] J. M. Lee. *Riemannian Manifolds: An Introduction to Curvature*, volume M. Springer, New York, 1997.

[27] J. M. Lee. *Introduction to Smooth Manifolds*. Springer, New York, 2003.

[28] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 1st edition, 2007.

[29] "E. Levina and P. Bickel". Maximum likelihood estimation of intrinsic dimension. *"Advances in NIPS"*, 17, 2005. "Vancouver Canada".

[30] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[31] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley, 2 edition, 1999.

[32] Marina Meila and William Pentney. "clustering by weighted cuts in directed graphs". *Proceedings of the 2007 SIAM International Conference on Data Mining*, 2007.

[33] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*, January 2013.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, 2013.

[35] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[36] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

[37] Boaz Nadler, Stephane Lafon, Ronald Coifman, and Ioannis Kevrekidis. Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 955–962, Cambridge, MA, 2006. MIT Press.

[38] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

[39] Umut Ozertem and Deniz Erdogmus. Locally defined principle curves and surfaces. *Journal of Machine Learning Research*, 12:1249–1286, 2011.

[40] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] D. Perraul-Joncas and M. Meila. Non-linear dimensionality reduction: Riemannian metric estimation and the problem of geometric discovery. *ArXiv e-prints*, May 2013.

[42] Dominique Perraul-Joncas and Marina Meila. Non-linear dimention reduction: Riemannian metric estimation and the problem of geometric recovery. *arXiv:1305.7255v1*, 2013.

[43] D. Perrault-Joncas and M. Meila. Improved graph Laplacian via geometric self-consistency. *ArXiv e-prints*, May 2014.

[44] Mary A. Rohrdanz, Wenwei Zheng, Maggioni Mauro, and Cecelia Clementi. "determination of reaction coordinates via locally scaled diffusion map". *The Journal of Chemical Physics*, 2011.

[45] S. Rosenberg. *The Laplacian on a Riemannian Manifold*. Cambridge University Press, 1997.

[46] Sam Roweis and Lawrence Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[47] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature Communications*, 8:13890, January 2017.

[48] A. Singer. From graph to manifold laplacian: the convergence rate. *Applied and Computational Harmonic Analysis*, 21(1):128–134, 2006.

[49] M. A. Strauss, D. H. Weinberg, R. H. Lupton, V. K. Narayanan, J. Annis, M. Bernardi, M. Blanton, S. Burles, A. J. Connolly, J. Dalcanton, M. Doi, D. Eisenstein, J. A. Frieman, M. Fukugita, J. E. Gunn, Ž. Ivezić, S. Kent, R. S. J. Kim, G. R. Knapp, R. G. Kron, J. A. Munn, H. J. Newberg, R. C. Nichol, S. Okamura, T. R. Quinn, M. W. Richmond, D. J. Schlegel, K. Shimasaku, M. SubbaRao, A. S. Szalay, D. Vanden Berk, M. S. Vogeley, B. Yanny, N. Yasuda, D. G. York, and I. Zehavi. Spectroscopic Target Selection in the Sloan Digital Sky Survey: The Main Galaxy Sample. *The Astronomical Journal*, 124:1810–1824, September 2002.

[50] Ameet Talwalkar, Sanjiv Kumar, Mehryar Mohri, and Henry Rowley. Large-scale svd and manifold learning. *Journal of Machine Learning Research*, 14:3129–3152, 2013.

[51] J. Tenenbaum, V. deSilva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[52] D. Ting, L Huang, and M. I. Jordan. An analysis of the convergence of graph laplacians. In *ICML*, pages 1079–1086, 2010.

[53] L.J.P. van der Maaten and G.E. Hinton. "visualizing high-dimensional data using t-sne". *Journal of Machine Learning Research*, pages 2579–2605, 2008.

[54] L.J.P. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review. 2008.

[55] J. VanderPlas and A. Connolly. Reducing the Dimensionality of Data: Locally Linear Embedding of Sloan Galaxy Spectra. *Astronomical Journal*, 138:1365–1379, November 2009.

[56] K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70:77–90, 2006. 10.1007/s11263-005-4939-z.

[57] Jevin D. West, Jennifer Jacquet, Molly M. King, Shelley J. Correll, and Carl T. Bergstrom. The role of gender in scholarly authorship. *PLOS ONE*, 8(7):1–6, 07 2013.

[58] T. Wittman. Manifold learning matlab demo. http://www.math.umn.edu/~wittman/mani/, retrieved 07/2010, 2010.

[59] C. W. Yip, A. J. Connolly, A. S. Szalay, T. Budavári, M. SubbaRao, J. A. Frieman, R. C. Nichol, A. M. Hopkins, D. G. York, S. Okamura, J. Brinkmann, I. Csabai, A. R. Thakar, M. Fukugita, and Ž. Ivezić. Distributions of Galaxy Spectral Types in the Sloan Digital Sky Survey. *Astronomical Journal*, 128:585–609, August 2004.

[60] Junping Zhang, Stan Z. Li, and Jue Wang. *Manifold Learning and Applications in Recognition*, pages 281–300. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[61] Z. Zhang and H. Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM J. Scientific Computing*, 26(1):313–338, 2004.