UNIVERSITY OF CAPE TOWN

MASTERS THESIS

ISO-maps for Non-linear Dimension Reduction - Addressing Geometric and Numerical Issues Observed in Practice.

Author:
Francois EVERT

Supervisor:
Dr. Etienne PIENAAR

A thesis submitted in fulfillment of the requirements for the degree of Masters in Data Science

in the

Statistical Sciences

June 13, 2022


University of Cape Town • iYunivesithi yaseKapa • Universiteit van Kaapstad
SPES BONA

Declaration of Authorship

I, Francois EVERT, declare that this thesis titled, "ISO-maps for Non-linear Dimension Reduction - Addressing Geometric and Numerical Issues Observed in Practice." and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"The fear of the Lord is the beginning of knowledge: but fools despise wisdom and instruction."

Proverbs 1:7 (KJV)

Abstract

The abundance of high dimensional data has necessitated various approaches to dimensionality reduction. Many of these methods make simplifying assumptions about the variation structure of the data and permit approximating high-dimensional structures using only a few components. In the context of modern day machine learning applications, such assumptions can rarely be justified. To this end, Tenenbaum, de Silva, and Langford [2000] proposed an approach for approximating the underlying non-linear structure on which the variation occurs and such assumptions would be more tenable. This is achieved by first finding the non-linear underlying structure (manifold) in a high dimensional input space and then using the shortest distance matrix of the data points along this manifold to achieve non-linear dimensionality reduction. The authors proposed a 3 step approach : Step 1 - constructing a **distance graph** by defining the neighbours of each data point such that it is contained within one section of the manifold, Step 2 - approximating the **geodesic** distances between all points using the graph constructed in Step 1 and Step 3 - applying **Classical MDS** to this distance graph to achieve dimensionality reduction.

The **first objective** of this research paper is to demonstrate that the approach by Tenenbaum et al. [2000] struggles to detect the manifold under conditions where the stochastic components of the data dominates to the extent that the underlying manifold may be difficult to approximate reliably. We propose and explore possible solutions on how to effectively deal with the variability in the data, such that the distance graph reliably reflect the **geodesic** interpoint distances. In principle, this deals with departures from the assumptions in Step 1 (constructing distance graph) and the remediation thereof.

The **second objective**, assuming that one can reliably detect the manifold, relates to the often substantial computational overhead of the algorithm: Step 2 of the algorithm attempts to approximate geodesic distances on the manifold by traversing paths on the graph constructed in Step 1, more specifically, the **shortest** paths. We test and demonstrate an alternative strategy for approximating distances by relaxing the constraint that the **shortest** path needs to be found, substituting the condition that **any** (reasonable) path between points on the graph suffices for purposes of this algorithm.

Lastly we demonstrate how the proposed remediations/variations on the algorithm can be used to conduct dimensionality reduction on real-world data sets, and provide

some conclusions and possible future reseach topics.

Acknowledgements

I hereby would like to acknowledge the following people without whom this endeavor would not have been possible.

First and foremost my Creator for providing the breeding ground for all that brought this thesis to fruition.

I would also like to express immense gratitude to my supervisor, Dr. Etienne PIENAAR, for his guidance and patience through this journey. Also, I'd like to thank the lecturers and other staff at University of Cape Town for their dedication to the betterment of man.

I would also not have been able to set out on this journey without the generous financial support of Vodacom SA. Then there are the colleagues who convinced me to take this on, who believed in me and my manager at Vodacom who generously granted me time off to work on this thesis and who kept me honest about my commitment to this.

Last, but not least, there are my friends and family, especially my two boys who showed a keen interest in seeing me succeed.

To all of you, I would have not have been here without your support, I remain forever in you debt.

"The older I get, the surer I am that I'm not running the show." — Leonard Cohen

Contents

List of Figures

List of Tables

List of Abbreviations

PCA	**P**rincipal **C**omponent **A**nalysis
NMF	**N**on-negative **M**atrix **F**actorization
LDA	**L**inear **D**iscriminant **A**nalysis
GDA	**G**eneralized **D**iscriminant **A**nalysis
t-SNE	**t**-distributed **S**tochastic **N**eighbour **E**mbedding
MDS	**M**ulti **D**imentional **S**caling

Dedicated to all who believed in
me and pushed me to the end.
You know who you are . . .

1 Introduction

While the modern age has brought an abundance of data with observations being generated at an ever increasing rate (Fan, Han, and Liu [2014]) in virtually all fields of human endeavor allowing unprecedented potential for statistical application, it is not only the number of observations that has changed the statistical landscape but also the shape of data. In particular, where the feature dimensions of data are concerned, significant challenges arise in the interface between models - the mathematical abstractions on which statistical analysis is built - and the recorded observations. This follows since, in the traditional modelling sphere, where simple linear models are used, interpretation becomes vague and the potential for spurious detection of relationships increases. Furthermore, in the 'modern' approach, using machine learning techniques, the number of parameters in such models depend on the input dimensions and, consequently, model complexity is implicit and the likelihood of overfitting increases (James, Witten, Hastie, and Tibshirani [2014]).

Although, in principle, managing model complexity to avoid overfitting data is not a new problem by any means, with well established procedures such as Principal component analysis (PCA), Kernel PCA and Graph-based kernel PCA, Non-negative matrix factorisation (NMF), Linear discriminant analysis (LDA), Generalised discriminant analysis (GDA), T-distributed Stochastic Neighbour Embedding (t-SNE) and Uniform manifold approximation and projection (UMAP), one is always confronted with departures from the assumptions on which such statistical probes operates. To this end Tenenbaum, de Silva, and Langford [2000] proposed a three step approach to finding the manifold in a high dimensional input space and then using the shortest distance matrix of the data points along the manifold to achieve non-linear dimensionality reduction. The first step focus on obtaining the neighbours of each data point, step two creates a matrix of geodesic distances (the shortest distance between two points on a surface, object or manifold, i.e. not the direct distance) between all points in the data space and the third step applies classical Multi Dimentional Scaling (MDS) to this distance matrix to best preserve the intrinsic geometry of the non-linear dataset. Although the algorithm presents a significant leap in providing an interface between dimensionality reduction techniques and non-linear structures, it too is subject to scrutiny of the assumptions on which it is built and the subsequent pragmatic issues that result therefrom.

When attempting dimensionality reduction in a (high dimensional) input space and considering the possibility of hidden non-linear data structures, the concept of a

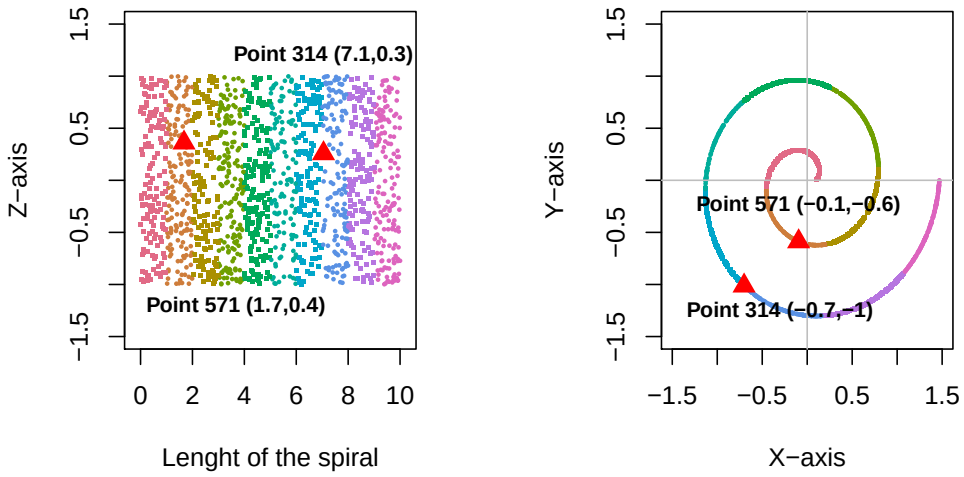


FIGURE 1.1: Original rectangular flat surface & 3-dimentional Swiss-Roll with no orthogonal variance - viewed along the z-axis

manifold emerges. A manifold can be defined as a subset of the n-dimensional input space, where each point on the surface of the manifold is surrounded by a flat surface, such that there are no sudden or step changes or sharp edges. If a manifold exists in the input space, such that (all) the data points live on or near the manifold and it can be detected and described, it would greatly enhance our understanding of the data and ease the visualisation thereof. Indeed, in statistical contexts involving modelling a response as a function of a set of inputs, one assumes that the position of a point in the input space is informative for purposes of predicting the response. Similarly, close proximity of two points implies similar predictions. In consequence, where such points lie on or near a manifold, position and proximity would be more accurately defined relative to the manifold than the origin of the space in which it lies. This is the key insight behind the manifold learning methodology. Each data point can then be described relative to each other using the manifold as the non-linear space through which to do so and as such, removing the non-linearity associated with the manifold.

As an expository example of such a configuration, consider Figure 1.1 which shows how a flat rectangularly dispersed set of data points on the left, is *rolled up* to create the "Swiss-Roll" on the right, here viewed co-axially along the Z-axis which spans the width of the Swiss-Roll. This Swiss-Roll forms the **manifold** which will be used in the rest of the analysis and evaluation. Each figure contains two highlighted data points, indicated with red triangles, which are the feature coordinates in a statistical learning problem, and will be used to show the various impacts of different methodologies. *(Refer to Appendix A for sample R code on how the Swiss-Roll is generated.)*

In the **first** of the three steps of the manifold learning methodology, manifold detection relies on the assumption that neighbouring data points in the high dimensional input space are also neighbours on the manifold. This assumption, already eroded by the curse of dimensionality, is further challenged by the introduction of **variance**, in

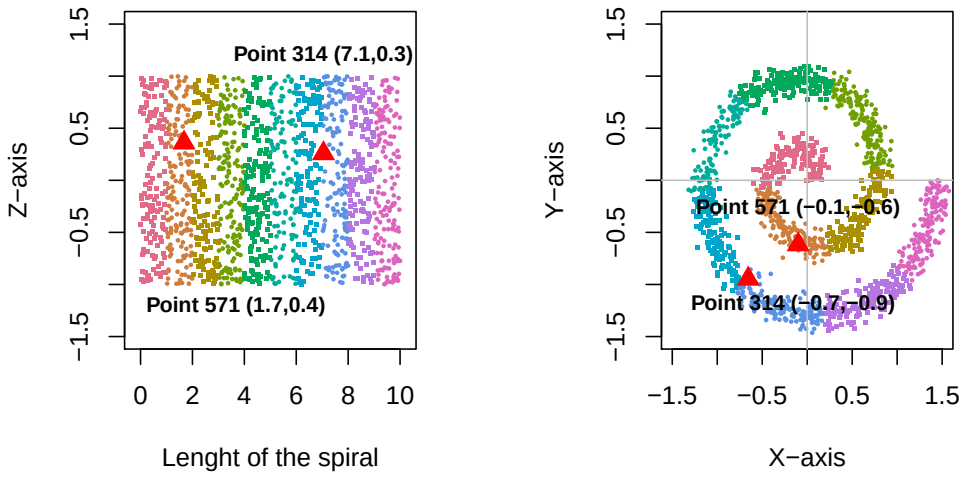


FIGURE 1.2: Original rectangular flat surface & 3-dimentional Swiss-Roll **with** orthogonal variance - viewed along the z-axis

particular variance which is **orthogonal** to the manifold. Closer analysis will show false manifold detection under these conditions and the devastating effect this will have on dimensionality reduction. Figure 1.2 shows the original flat rectangularly dispersed set of data points on the left and 3-dimentional Swiss-Roll **with** orthogonal variance - here viewed co-axially along the Z-axis which spans the width of the Swiss-Roll.

The **second** step in the proposed methodology calculates the shortest geodesic path between all points in the input space. And while this pose no problem for the small examples described in the article, finding the **shortest** path in a large dataset can be computationally intensive. Further analysis will challenge the notion that the **shortest** path is required and rather postulate that applying classical MDS to a matrix with **any** path between all points on the manifold will still result in positive dimensionality reduction, thus eliminating the need for costly shortest path algorithms.

The **third** step entails the simple application of Classical MDS to the derived distance matrix which follows the manifold, thus preserving the Euclidean distances along the manifold, i.e. geodesic distances. In this analysis, applying MDS on the distance graph effectively "un-rolls" the Swiss-Roll, back to its original flat rectangular set of data points.

The remainder of this dissertation is outlined as follows: In the following chapter (Chapter 2) we explore the existing literature. Subsequently (Chapter 3), we detail the technical aspects and notation of the existing methodology and proposed methods. In Chapter 4, we introduce variance on the dataset and explore the effect thereof on the exisiting methodology. Chapter 5 proposes various remediations and evaluate its relative performance on the original Swiss-roll and the well known MNIST datasets. Chapter 6 explores the computational gains which could be obtained from variations

in the path selection process. Finally (Chapter 7), we apply remediation to real-world datasets and present some concluding remarks.

For readers unfamiliar with these methods and/or new terminology introduced in the present dissertation, refer to Appendix F for a summary.

2 Literature Review

The theory and implementation of non-linear dimensionality reduction has been well researched and documented, with increased frequency over the last 20 years, citing among many others Tasoulis, Pavlidis, and Roos [2020], Orsenigo and Vercellis [2013], Demers and Cottrell Y [1993] and Van Der Maaten, Postma, and Van den Herik [2009]. The literature describes a move from linear techniques such as Principal Components Analysis (PCA), factor analysis and classical scaling, to non-linear techniques such as Kernel PCA, neighbour, Maximum Variance Unfolding, diffusion maps, Locally Linear Embedding, Laplacian Eigenmaps, Hessian LLE, Local Tangent Space Analysis, Sammon mapping, multilayer autoencoders, Locally Linear Coordination and manifold charting. The successful implementation of these techniques in real-world scenarios has also been widely documented, e.g. Yang, Wang, Ding, An, and Alterovitz [2017] in the field of biology and Orsenigo and Vercellis [2013] in the banking industry.

In particular, this thesis focuses on the neighbour technique as put forward by Tenenbaum, de Silva, and Langford [2000]. Their approach (which we replicate in this thesis) is step-wise demonstrated using a Swiss-roll dataset, with further application in facial, hand and handwritten digits recognition yielding positive results.

However, Balasubramanian and Schwartz [2002] commented on the stability (amongst other criticisms) of this approach, stating that: "*this approach is topologically unstable and can only be used after careful preprocessing of the data*". In the presence of variability (we assume that the data points are not generated deterministically, but rather stochastically), in particular such that are orthogonal to the manifold, the construction of shortest paths fail to find the true geodesic distances by following "short-circuits" or "jumps" between different sections of the manifold. The authors elaborate on, and demonstrate the susceptibility of this procedure to incorrect manifold detection due to unsatisfactory neighbourhood definition, leading to catastrophic failure in accurately performing dimensionality reduction. According to the authors, the procedure hinges on selecting a **neighbourhood** that is "just right" (in either of K-nearest or epsilon-neighbourhood configurations) which requires "*a priori* information" about the manifold.

In their response to the comments by Balasubramanian and Schwartz [2002], Tenenbaum et al. maintains that it would be inaccurate to characterise the procedure as topologically unstable. The authors state that an appropriate neighbourhood size can

be chosen for the case which Balasubramanian and Schwartz [2002] use to highlight the issue. The authors go on to elaborate on noise tolerances for various scenarios with varying dataset sizes and degrees of curvature of the manifold. It does, however, imply that, since there are noise tolerances, there are indeed datasets which exceeds these tolerances and as such would not yield trustworthy results. The evaluation thereof would thus, to some extent, rely on closer evaluation of the results by subject matter experts.

Van Der Maaten, Postma, and Van den Herik [2009] build on the comments by Balasubramanian and Schwartz [2002] about the "topological instability" of the neighbour procedure. The authors list three weaknesses, starting with the instability as highlighted in Balasubramanian and Schwartz [2002]. Secondly, the authors indicate that the procedure does not fare well when the manifold contains "holes" or rather, regions in the feature space void of observations, e.g. coordinates which cannot be observed or do not exist in the data generating process. However, while there are techniques for handling manifolds with essential loops, Lee and Verleysen [2005], it is deemed outside the scope of this thesis, thus restricting the datasets with no non-contractible loops, e.g. the torus was excluded as it introduces more complexity which should be dealt with separately. Thirdly, the authors state that the neighbour procedure could fail, should the manifold be nonconvex and again, as in the case of essential loops, such manifolds are excluded from this thesis and should be dealt with separately.

A fourth weakness (in addition to those mentioned above) is the resource intensiveness of the shortest path algorithm. The two algorithms proposed by Tenenbaum et al. [2000] for finding the shortest path are Floyd's algorithm as found in Floyd [1962] and Dijkstra's algorithm as found in Dijkstra [1959]. These algorithms action upper limits of $O(V^3)$ and $O(V^2)$ respectively, which could lead to a increased computational overhead for large datasets. This weakness is further addressed in Chapter 6). Though we do not claim to improve on the order of the path search algorithm, we do attempt to demonstrate that computational gains can be made by relaxing constraints on the paths calculated.

An article by Mahwish Yousaf [2020] explores the use of a Randomised KD-tree as a first algorithm, which refines the distance graph. The second step is then the application of a NN-Descent algorithm on this refined distance graph. It yielded extremely good results of above 99% (100% in most cases) accuracy, where ISOmap averaged a mere 55% accuracy.

While the neighbour approach has been implemented by Oksanen, Blanchet, Friendly, Kindt, Legendre, McGlinn, Minchin, O'Hara, Simpson, Solymos, Stevens, Szoecs, and Wagner [2020] in the R CRAN as part of the "vegan" package, as a function called "isomap", for the purpose of this thesis, personally developed code and the "allshortestpath" function from the "e1071" package (Meyer, Dimitriadou, Hornik,

Weingessel, and Leisch [2020]) was used to evaluate various options to address the weaknesses of the neighbour procedure. Additionally, the "redDim" package (Kraemer, Reichstein, and Mahecha [2018]) was used for additional verification of remediation in Chapter 7.

3 A Global Geometric Framework for Non-linear Dimensionality Reduction by Tenenbaum et al. [2000]

Tenenbaum et al. [2000] propose an algorithm for conducting manifold-learning in high-dimensional input spaces. The algorithm consists of three distinct steps which are subsequently detailed below, but the premise of the algorithm can be summarised as follows: extract the underlying manifold on (or close to) which observations are said to occur, measure the position of observations with reference to their position on (or proximate to) the manifold and conduct standard dimension reduction techniques (MDS, PCA, etc.) based on distances measured relative to the manifold.

In practice, the procedure is carried out as follows and we demonstrate the technique graphically using the canonical example of the so-called "Swiss-Roll": Given a set of N data points $\{\mathbf{x}_i' : i = 1, 2, \dots N\}$ in a high dimensional input space $\mathcal{X}$. Let the dimension of the input space be p (hence the dimension of $\mathbf{x}_i$ is $p \times 1$).

In the **first step** of the Tenenbaum algorithm, neighbour identification is achieved by employing K-nearest neighbour, or by defining a neighbourhood radius ϵ such that only data points closer (euclidean distance) than ϵ from any given point is considered a neighbour. That is, define for each pair (i, j) an index set

$$\mathcal{I}_{ij} = \{(i, j) : 1 < \text{rank}(d_X(i, j)) < k + 1\}$$

where rank here means smallest to largest (since when $j = i$ $d_X(i, j) = 0$, rank 1 will correspond to the same observation), or

$$\mathcal{I}_{ij} = \{(i, j) : d_X(i, j) < \epsilon, i \neq j\}$$

and an edge-set (the relevance of "edge" will become apparent shortly)

$$\mathcal{E}_{ij} = \{d_X(r, s) : r, s \in \mathcal{I}_{ij}\},$$

where $d_X(i, j)$ represents the euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. An undirected

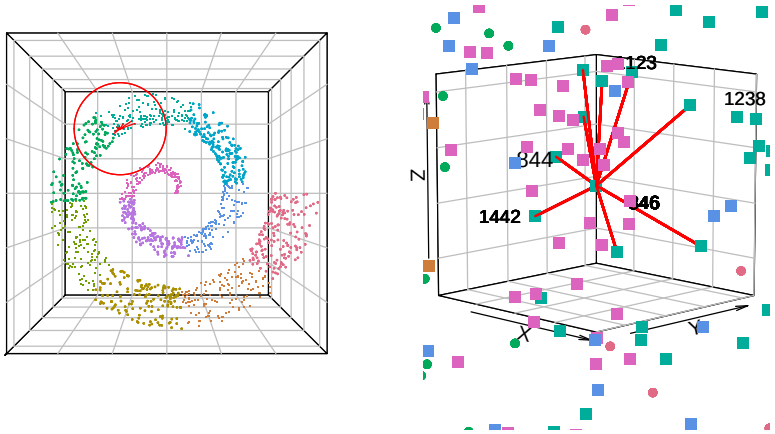


FIGURE 3.1: Scatterplot of Swiss-Roll depicting the nine nearest neighbours to point ***Pt:210***

graph G is then constructed by connecting all elements in all index sets and setting the edge-lengths equal to the euclidean distances between observations (captured in the edge-set $\mathcal{E}$). Constructing graph G then concludes the first step, thereby essentially approximating the underlying manifold.

Figure 3.1 shows a 3-dimentional scatterplot of a Swiss-Roll, highlighting one point (chosen at random, for illustrative purposes) (***Pt:210***) with line segments to its nine nearest neighbours on the left. On the right is a closer look at point ***Pt:210*** and its connections to its neighbours.

Figure 3.2 shows a 3-dimentional scatterplot of the Swiss-Roll with all nine nearest neighbours of all points connected, viewed along the z-axis.

The **second step** uses graph G to find the paths on the graph which are shortest among all possible paths from one vertex to another. To this end, various algorithms exist, including Floyd's algorithm (Floyd [1962], Grama et al. [2003]), Dijkstra's sequential single-source shortest paths algorithm (Grama et al. [2003]) and Dijkstra's all-pairs shortest paths algorithm (Grama et al. [2003]). In this Swiss-Roll example, Floyd's algorithm is employed (it is not clear that there is preference for any one path finding algorithm over another, apart from possible computational gains) by initialising a distance matrix d_G such that for all points i and its neighbour j (which are directly linked), $d_G(i,j) = d_X(i,j)$, while all other sets of points are assigned an infinite distance i.e. $d_G(i,j) = \infty$. An iterative process evaluates and replaces each distance $d_G(i,j)$ by $\min\{d_G(i,j), d_G(i,k) + d_G(k,j)\}$ for each $k \in 1..N$. The resultant matrix of distances $D_G = \{d_G(i,j)\}$ contains the shortest **geodesic** distance between each pair (i,j) in G which concludes the second step.

Figure 3.3 shows the shortest path between points Pt:314 and Pt:571 as determined by Floyd's algorithm.

Table 3.1 shows the nearest neighbours of the points in the shortest path between points Pt:314 and Pt:571 as determined by Floyd's' algorithm. The neighbour selected

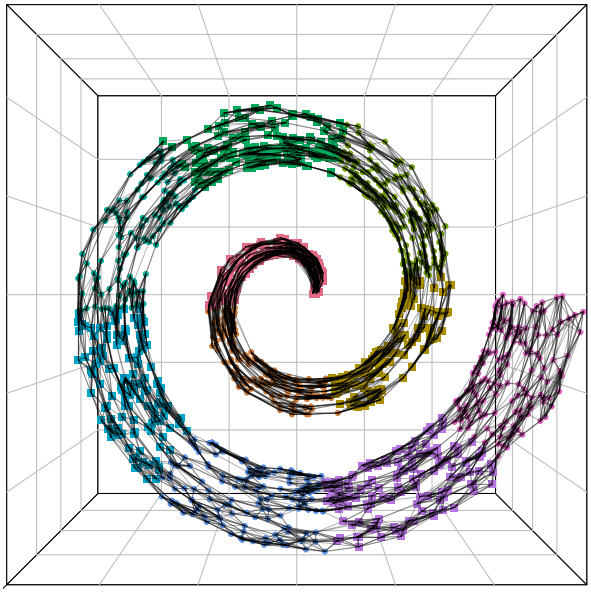


FIGURE 3.2: Scatterplot of Swiss-Roll depicting all nearest neighbours connected

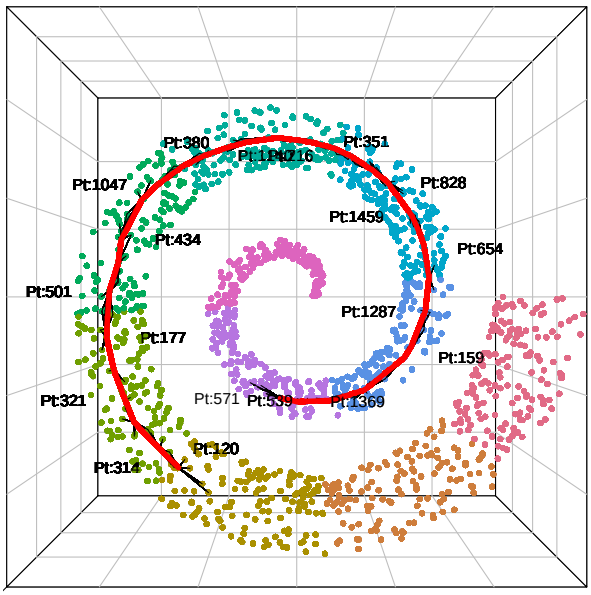


FIGURE 3.3: Scatterplot of Swiss-Roll depicting the shortest path between two points: ***Pt:314 and Pt:571***

TABLE 3.1: Table showing the nearest neighbours of the points in the path: ***Pt:314 and Pt:571***

POINTS		**NN1**	**NN2**	**NN3**	**NN4**	**NN5**	**NN6**	**NN7**	**NN8**	**NN9**
314	43	834	1395	**120**	213	577	96	210	1460	
120		1395	577	369	142	314	834	43	503	**1479**
...		...	...	...	...	...	...	...	...	...
1369		156	1449	**571**	908	749	490	1352	603	539
571		1369	156	603	147	981	182	1449	338	1492

TABLE 3.2: Table showing the edge distances to the nearest neighbours of the points in the path: ***Pt:314 and Pt:571***

POINTS	**Edge1**	**Edge2**	**Edge3**	**Edge4**	**Edge5**	**Edge6**	**Edge7**	**Edge8**	**Edge9**
314	2.154	1.885	2.188	**1.174**	1.419	2.44	0.652	0.939	1.49
120	2.12	1.775	2.268	1.25	1.549	2.503	0.563	0.798	**1.449**
...	...	...	...	...	...	...	...	...	...
1369	1.535	1.787	1.47	0.455	1.003	1.767	1.123	1.207	1.081
571	1.548	1.752	**1.569**	0.5	1.049	1.862	1.076	1.109	1.019

to form part of the shortest path is shown in **bold**. Only the first and last two points are shown; there are a total of thirty-five points in this path.

Table 3.2 shows the edge lengths of the nearest neighbours of the points in the shortest path between points Pt:314 and Pt:571 as determined by Floyd's algorithm. The edge lengths selected to form part of the shortest path are shown in **bold**. Only the first and last two points are shown, there are a total of thirty-five points in this path.

In the third step, classical MDS is applied to the graph-distance matrix $D_G = \{d_G(i,j)\}$. This produces a d-dimensional Euclidean space Y which best preserves the estimated intrinsic geometry of the manifold. However, two pitfalls arise when choosing the number of nearest neighbours (or radius within which to include data points). Should the number of nearest neighbours be too large (or the inclusion radius too big), the risk of including data points which are not a short geodesic distance from the given original data point increases. On the other hand, should the number of nearest neighbours be too small (or the inclusion radius too short), the risk of there not being a shortest geodesic distance between all data points increases.

Figure 3.4 shows the original square flat surface which was used to create the Swiss-Roll used in this analysis on the left and on the right, the 2-dimentional representation of the 3-dimentional Swiss-Roll (with no orthogonal variance). It is clear from this figure that the procedure effectively "un-rolled" the Swiss-roll, preserving the interpoint distances of the original flat rectangular surface.

It should be noted that the approach proposed by Tenenbaum et al. [2000] has several possible weakneses, including stability of the algorithm, catastrophic failure in the presence of orthogonal variance, sensitivity to parameter selection, reliance on *a priori* knowledge of the dataset (Balasubramanian and Schwartz [2002]), failure to deal well with complexities when the manifold contains "holes" (regions void of observations)

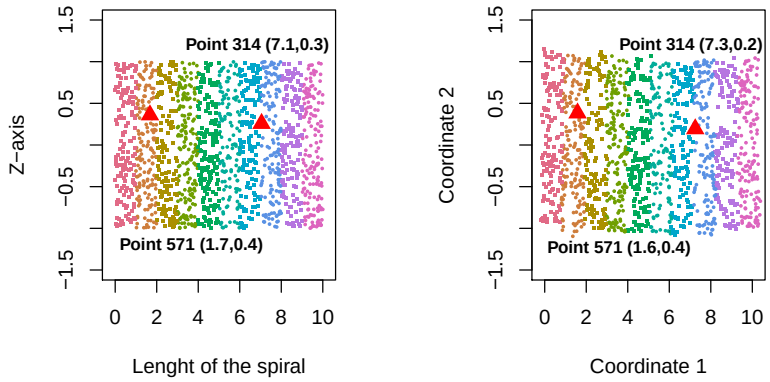


FIGURE 3.4: Flat surface rolled into Swiss-Roll vs. 2-dimentional MDS representation of the Swiss-Roll with no variance

(Van Der Maaten et al. [2009]) and excessive computational overhead during path generation in datasets with many observations. We will be exploring and proposing remedies to these with particular focus on addressing concerns related to varaince in Chapter 4, while in Chapter 5 we explore and propose an approach to address the computational load in path selection.

4 Variance in the Dataset

While the solution to finding the shortest geodesic path between two points on a manifold proposed by Tenenbaum et al. [2000] proved effective in the simple case of a Swiss-Roll with zero orthogonal variance or where observations are close to or on the manifold. This implicitly assumes a high 'signal-to-noise' ratio. Now, the use of such technology is obviously predicated on statisticians checking the validity of such assumptions before applying such machinery, it is still plausible that data which is suspected to have some underlying non-linear manifold is subject to statistical variation (i.e. observations are not generated deterministically on the manifold surface but rather proximate to it). Indeed, this can have a significant impact on the performance of the algorithm. For purposes of testing the performance of the proposed method, we demonstrate by way of one possible configuration for variation on the manifold, namely, orthogonal variance. Adding orthogonal variance to the Swiss-Roll dataset is a simple statistical configuration. (Other options are variation in the shape of the manifold (also refered to as deterministic noise) and ambiguity in classification bounds of discrete valued responses.) Under this configuration a normally distributed random variable is added to the spiral radius during the Swiss-roll generation process. Care was taken to distribute the data points evenly along the spiral, ensuring that a constant variance is applied over the length of the spiral. This is done to avoid heteroscedasticity, which, if present, would introduce even more complexity in finding a homogenous solution.

As in the zero variance case (i.e. where the observations occur ON the manifold), the nearest neighbours to each data point is identified. However, as can be seen from Figure 4.1, there are cases where the data points are no longer in close proximity of the manifold. This leads to different sections of the manifold being deemed neighbours, e.g. point ***846*** has a point ***463*** as a neighbour, but these two points belong to different sections of the manifold. Figure 4.1 shows a 3-dimentional scatterplot of a Swiss-Roll with variance orthogonal to the manifold, highlighting one point (***846***) with line segments to its nine nearest neighbours on the left. On the right is a closer look at point ***846*** and its connections to its neighbours. Figure 4.2 shows a 3-dimentional scatterplot of the Swiss-Roll with orthogonal variance and all nine nearest neighbours of all points connected, viewed along the z-axis. The interconnections which spans different sections of the Swiss-Roll are highlighted as thick red lines.

Scenarios where the neighbour of a point belongs to a different section of the manifold introduce inaccuracies (poor approximations) when determining the shortest geodesic

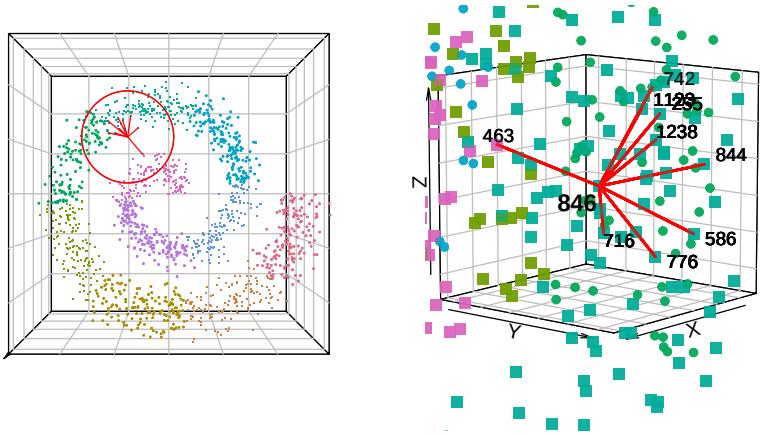


FIGURE 4.1: Scatterplot of Swiss-Roll with orthogonal variance, depicting the nine nearest neighbours to point ***846***

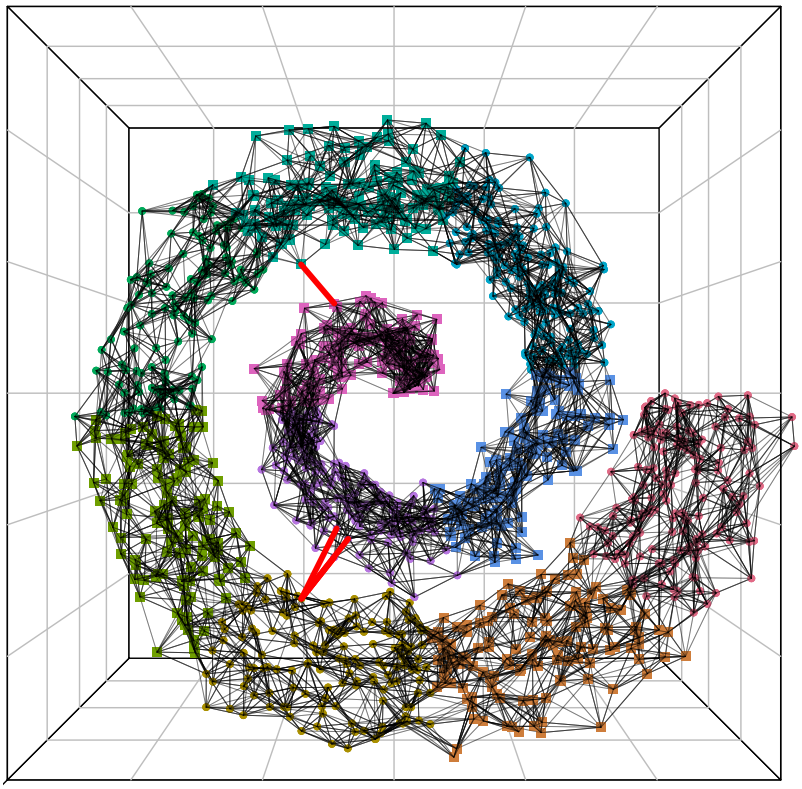


FIGURE 4.2: Scatterplot of Swiss-Roll depicting all nearest neighbours connected

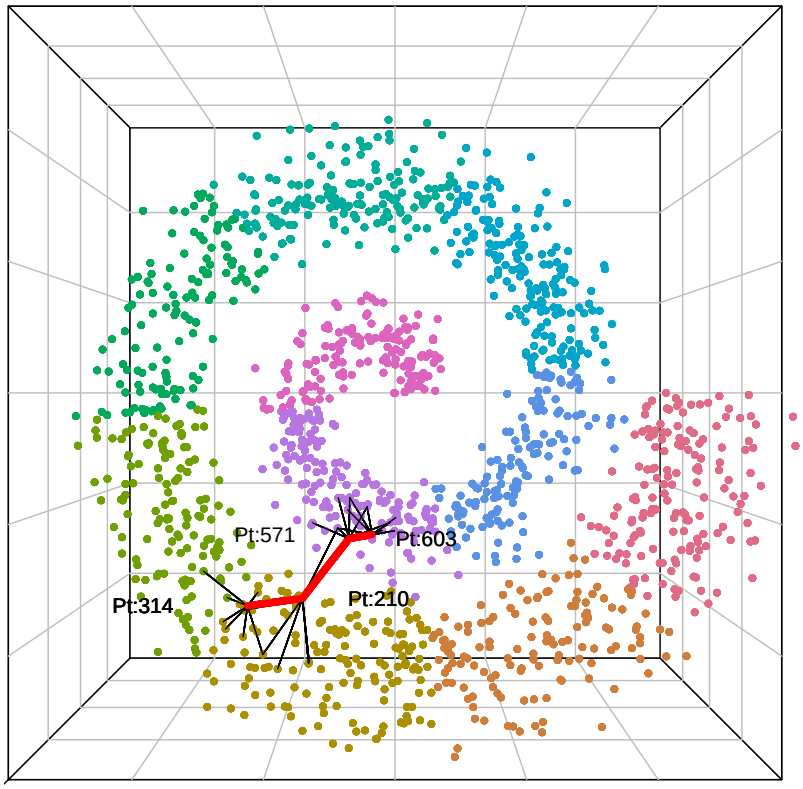


FIGURE 4.3: Scatterplot of Swiss-Roll depicting the incorrect shortest geodesic path between two points: ***Pt:314 and Pt:571***

path between points. The path no longer follows the manifold, but "jumps" between sections thereby creating the false impression that the points are closer to each other than what is the case when following the manifold.

Figure 4.3 shows the shortest geodesic path between points 500 and 800 as determined by Floyd's algorithm (Floyd [1962]). It clearly shows the jump between the layers of the Swiss-Roll.

Table 4.1 shows the nearest neighbours of the points in the shortest path between points Pt:314 and Pt:571 as determined by Floyd's algorithm (Floyd [1962]). The neighbour selected to form part of the shortest path is shown in **bold**.

Table 4.2 shows the edge lengths of the nearest neighbours of the points in the shortest path between points Pt:314 and Pt:571 as determined by Floyd's algorithm (Floyd [1962]). The edge lengths selected to form part of the shortest path is shown in **bold**. Unlike tables 3.1 and 3.2, in the presence of orthogonal variance, the path reduces to just four points (actual path is thirty-five points), having jumped between different sections of the Swiss-roll.

When classical MDS is applied to this skewed graph-distance matrix, the true geodesic distances are misrepresented, in this example, leading to a skewed 2-dimensional

TABLE 4.1: Table showing the nearest neighbours of the points in the path: ***Pt:314 and Pt:571***

POINTS	NN1	NN2	NN3	NN4	NN5	NN6	NN7	NN8	NN9
314	43	1395	834	577	213	96	1460	**210**	369
210	314	562	43	1264	164	406	213	147	**603**
603	147	**571**	981	1492	182	400	156	1056	1369
571	1369	156	603	147	981	182	338	908	1056

TABLE 4.2: Table showing the edge distances to the nearest neighbours of the points in the path: ***Pt:314 and Pt:571***

POINTS	Edge1	Edge2	Edge3	Edge4	Edge5	Edge6	Edge7	Edge8	Edge9
314	2.019	1.803	2.113	1.102	1.374	2.356	0.647	**0.926**	1.47
210	1.879	1.843	1.888	0.843	1.13	2.162	0.841	1.046	**1.357**
603	1.581	**1.749**	1.68	0.528	1.034	1.971	1.05	1.071	1.078
571	1.522	1.733	1.56	0.479	1.01	1.848	1.072	1.159	1.126

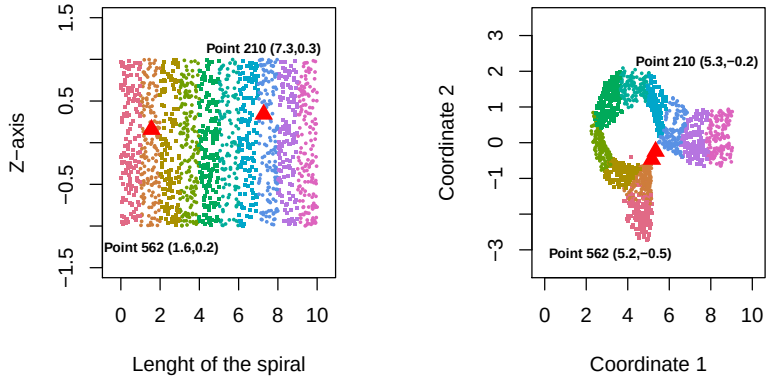


FIGURE 4.4: Flat surface rolled into Swiss-Roll vs. 2-dimentional MDS representation of the Swiss-Roll with variance

representation of the manifold.

Figure 4.4 shows the original square flat surface which was used to create the Swiss-Roll used in this analysis on the left, with points 210 and 562 highlighted with red triangles. On the right, the 2-dimentional representation of the 3-dimentional Swiss-Roll (with orthogonal variance) is depicted, showing points 210 and 562 much closer to each other than it was in the original flat surface on the left.

5 Proposed Remediation in the Presence of Stochastic Variation

As demonstrated in Chapter 4, in datasets where the stochastic components of the data dominate, manifold detection is compromised, resulting in a misrepresentation of the actual geodesic distances between data points. In order to remedy this situation, various approaches in identification, modification and/or removal of data points that skew the true shape of the manifold are investigated and tested. These approaches were tested on a Swiss-Roll dataset generated for purposes of the present paper, and extracts of the R code can be found on GitHub (Evert)

The typesetting used in reference to the dataset and its properties is listed below:

$\mathbf{x}_i$	i^{th} **data point** (as vector)
$\mathbf{x}_{ij}$	vector from data point i to data point j
$\dot{\mathbf{x}}_i$	i^{th} **new or altered** data point (as vector)
$d_X(ij)$	**distance** from data point i to data point j
D_G	matrix of **geodesic distances**
$\mathcal{S}$	a (sub-)**set** (of data points)
$\dot{\mathcal{S}}$	an altered (sub-)**set** (of data points)
N	**number** of data points (usually of the **full set of data points**)
K	**number** of data points in a subset (usually in reference to **nearest neighbours**)
α, ϵ	parameters set by the statistician

5.1 Mean of K-Nearest Neighbours

A very basic approach to dealing with outliers in the dataset is to redefine each data point as the mean of each dimension (Cartesian coordinates in the Swiss-Roll example) of its neighbours, such that

$$\dot{\mathbf{x}}_i = \frac{1}{K} \sum_{j \in \mathcal{S}} \mathbf{x}_j$$

where $\mathcal{S} = \{k \in 1, 2, \ldots N : 1 < \text{rank}(\ldots) < K + 1\}$. Although the ISOmap algorithm uses nearest neighbours, this is a transformation which is applied before the ISOmap algorithm is applied, and each data point is modified in turn. The reasoning behind this approach is that, in the case where a particular data point has neighbours in more than one area of the manifold, usually an outlying coordinate and it has more

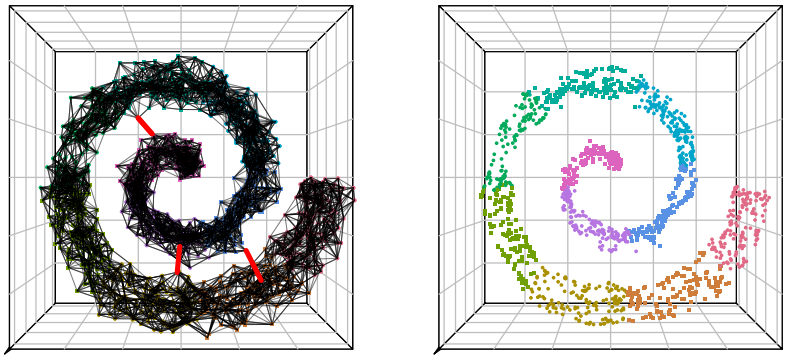


FIGURE 5.1: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section Mean of K-Nearest Neighbours) variance (right)

neighbours in one area of the manifold than in the other area(s), the weight of those (majority) neighbours would "pull" the data point closer to that particular area, thereby reducing the likelihood of it being an outlier. Note some potential issues with this: for example, that two observations could have the same neighbours in which case the transformed variables will have the same coordinate. This could be remedied by weighting the observation coordinate in:

$$\alpha x_i + \frac{(1-\alpha)}{K} \sum_j x_j$$

where $\alpha \in [0,1]$ and $x_\cdot$ are feature coordinates, thus introducing an additional control parameter with which the data perturbation can be altered between simply returning the original point and the coordinate produced by averaging nearest neighbours.

The 3-step procedure from Tenenbaum et al. [2000] is then followed on this altered dataset where the variability in the data has been reduced, see Figure 5.1. While this approach yielded some positive results in terms of reducing outliers (see Figure 5.2), the main drawback is that **all** data points are altered in the process, thus producing a somewhat skewed view of the original data and to some extent, the data points are clustered together, leaving "holes" in the resultant reduced dimension output, as can be seen in Figure 5.3.

5.2 Mean of K-Nearest Neighbours with Limit

A refinement of the previous approach is to similarly redefine each data point as the mean of each dimension of its neighbours,

$$\dot{\mathbf{x}}_i = \frac{1}{\dot{K}} \sum_{j \in \dot{\mathcal{S}}} \mathbf{x}_j$$

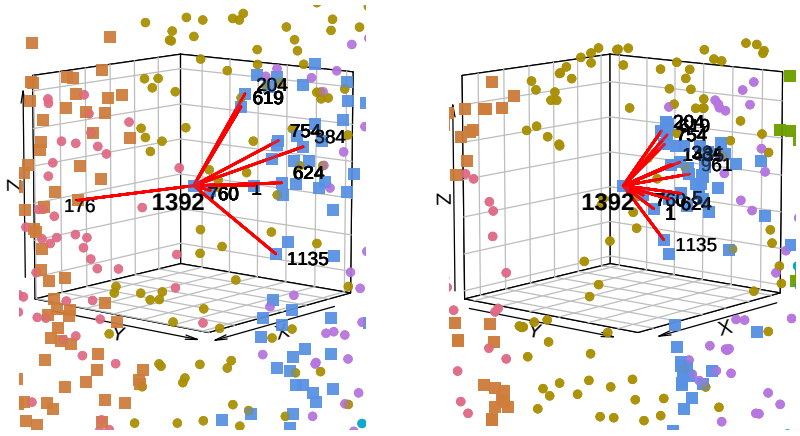


FIGURE 5.2: Scatterplot of Swiss-Roll with orthogonal variance, before (left) and after (right) (according to Section Mean of K-Nearest Neighbours, depicting the nine nearest neighbours to point ***846***

where $\dot{\mathcal{S}} = \{k \in 1, 2, \ldots N : 1 < \text{rank}(\ldots) < \dot{K} + 1\}$. This in-turn imposes selection on the neibouring coordinates used to represent the transformed input coordinate. The refinement lies in that the neighbour subset $\dot{\mathcal{S}}$ is limited such that the distance to each neighbour is less than a factor α of the mean Euclidean distance to all its neighbours, given by

$$||\mathbf{x}_i - \mathbf{x}_j|| < \alpha \times \frac{1}{K} \sum_{m \in \mathcal{S}} ||\mathbf{x}_i - \mathbf{x}_m||$$

where $\mathcal{S} = \{k \in 1, 2, \ldots N : 1 < \text{rank}(\ldots) < K + 1\}$. This is in line with the proposal of Tenenbaum et al. [2000] to consider only points closer than ϵ from any given point as a neighbour : $\mathcal{I}_{ij} = \{(i, j) : d_X(i, j) < \epsilon, i \neq j\}$, with the difference being that while the ϵ value is static, $\alpha \times \frac{1}{K} \sum_{m \in \mathcal{S}} ||\mathbf{x}_i - \mathbf{x}_m||$ is locally defined at each point, which aims to build in a level of robustness against the unwelcome effects of heteroscedastic components in the dataset. While this remediation can cause some observations to be closer to others (less variance), it might also cause a level of clustering, which decreases the probability of finding paths between all data points, as can be seen in the first test case in 7

This refinement (with $\alpha = 1.3$ as chosen by the researcher) yielded very similar results to Section 5.1, as seen in Figures 5.4 and 5.5. It is however expected to outperform (based on distance RMS ratio) the approach in Section 5.1 under conditions of heteroscedasticity.

5.3 Mean K-Nearest 1st and 2nd Level Neighbours

In the first step, as proposed by the authors, when determining the number of neighbours or the radius ϵ within which points are deemed neighbours, there is a tradeoff between selecting enough neighbours such that a shortest path exists between all points and not selecting too many neighbours such that it spans multiple

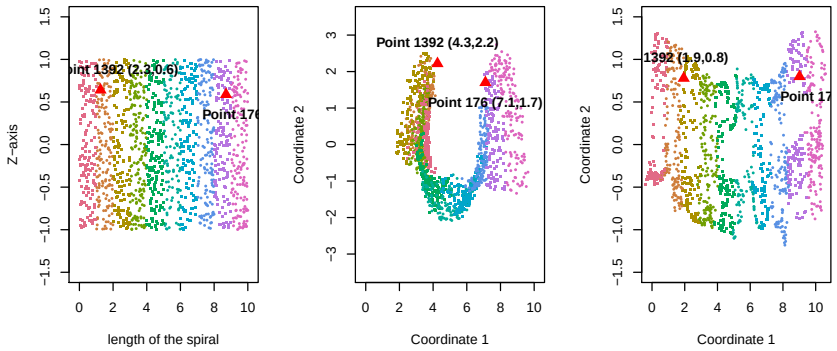


FIGURE 5.3: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Mean of K-Nearest Neighbours

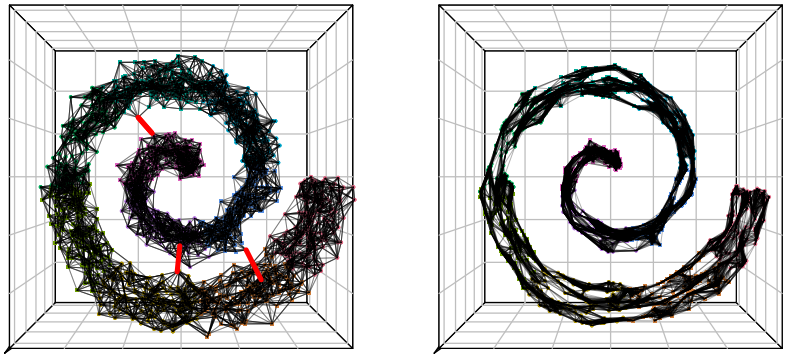


FIGURE 5.4: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section Mean of K-Nearest Neighbours with Limit variance (right)

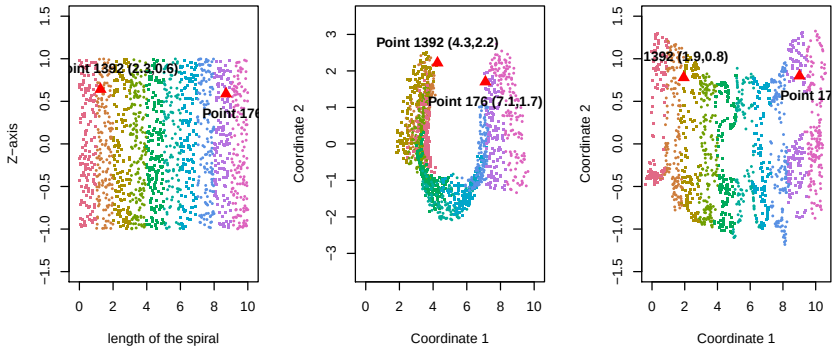


FIGURE 5.5: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Mean of K-Nearest Neighbours with Limit

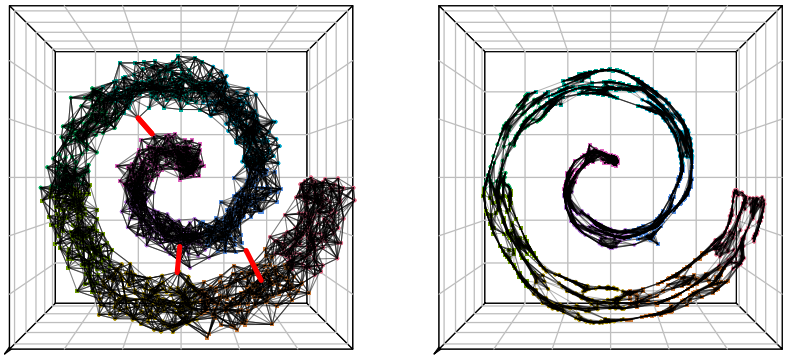


FIGURE 5.6: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section (Mean K-Nearest 1st and 2nd Level Neighbours) variance (right)

areas of the manifold. In an attempt to be more selective in the choice of neighbours, rather that just increasing N or ϵ, the neighbours of the primary set of neighbours of each data point, are also regarded as part of the neighbours pool. Once again, each data point is redefined as the mean of each dimension (Cartesian coordinates in the Swiss-Roll example) of this augmented subset

$$\dot{\mathbf{x}}_i = \frac{1}{\dot{K}} \sum_{j \in \dot{\mathcal{S}}} \mathbf{x}_j$$

where $\dot{\mathcal{S}} = \{k \in 1, 2, \ldots N : 1 < \text{rank}(\ldots) < \dot{K} + 1\}$. The net effect is that the variability in the dataset was greatly reduced, as can be seen in Figure 5.6.

Similar to 5.2, some level of success was achieved, with the final output being somewhat clustered (5.7).

5.4 Mean K-Nearest 1st and 2nd Level Neighbours with Limit

As in Sections 5.2 and 5.3, the subsection of points deemed neighbours can be expanded to include neighbours of neighbours, but also limited such that only data points closer than a factor of the mean neighbour of neighbour distances are considered. And, using this subset of data points, $\dot{\mathcal{S}}$, each data point is redefined as the mean of each dimension, such that

$$\dot{\mathbf{x}}_i = \frac{1}{\dot{K}} \sum_{j \in \dot{\mathcal{S}}} \mathbf{x}_j$$

where $\dot{\mathcal{S}} = \{k \in 1, 2, \ldots N : 1 < \text{rank}(\ldots) < \dot{K} + 1\}$ and

$$||\mathbf{x}_i - \mathbf{x}_j|| < \alpha \times \frac{1}{K} \sum_{m \in \dot{\mathcal{S}}} ||\mathbf{x}_i - \mathbf{x}_m||$$

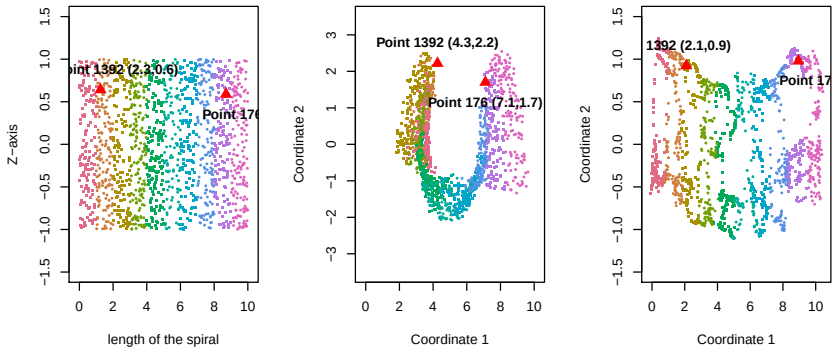


FIGURE 5.7: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Mean K-Nearest 1st and 2nd Level Neighbours

and α again chosen such that there the limited augmented neighbour subset spans only one area of the manifold while still providing a shortest path between all data points ($\alpha = 1.3$). Similarly to previous approaches, the variability in the dataset was greatly reduced, as can be seen in Figure 5.8 and even more pronounced clustering in the MDS output as can be seen in Figure 5.9.

5.5 Mean K-Nearest 1st and 2nd Level Neighbours with Dot Product Limit

Similar to the approach in Section 5.4, each data point is redefined as the mean of each dimension of its neighbours, and neighbours of neighbours,

$$\dot{\mathbf{x}}_i = \frac{1}{\dot{K}} \sum_{j \in \dot{\mathcal{S}}} \mathbf{x}_j$$

where $\dot{\mathcal{S}} = \{k \in 1, 2, \ldots N : 1 < \text{rank}(\ldots) < \dot{K} + 1\}$. The neighbour subset $\dot{\mathcal{S}}$ is defined such that only neighbours and neighbours of neighbours with a sufficiently large dot product between the vectors from the i^{th} point to each neighbour and the vector from the i^{th} point to its nearest neighbour

$$\mathbf{x}_{ij} \cdot \mathbf{x}_{ik}$$

where $\mathbf{x}_j$ is the nearest (1^{st}) neighbour to data point $\mathbf{x}_i$ and $\mathbf{x}_k$ is the k^{th} neighbour or neighbour of neighbours with $k \in 2, 3, \ldots N : 2 < \text{rank}(\ldots) < \dot{K} + 1\}$, are considered viable neighbours. The aim is to only consider the neighbours and neighbours of neighbours that resembles the **largest directional component** to the vector from each point to its nearest neighbour, thereby eliminating data points that has large orthogonal components to the primary neighbour vectors (from each point to its

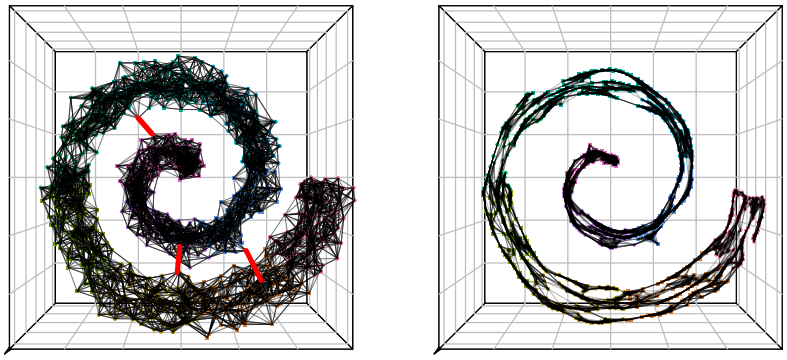


FIGURE 5.8: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section (Mean K-Nearest 1st and 2nd Level Neighbours with Limit) variance (right)

nearest neighbour). It should also be noted that because this methodology deals with an undirected distance graph, the **absolute value** of the dot product was used. In this example, 50% of the neighbours and neighbours of neighbours with the largest dot product were selected and it yielded promising results, with less distortion of the original dataset and less cluttering in the MDS output, as can be seen in Figures 5.10 and 5.11.

5.6 Distance to Plane

In this approach, it is assumed that the **plane** $\mathbf{P}_i$ defined by each point $\mathbf{x}_i$ and its two nearest neighbours $\mathbf{x}_j$ and $\mathbf{x}_k$ approximates the manifold at point i. The distance between this plane and each of the remaining neighbours gives an indication of how close that neighbour approximates the manifold. Observations for which distance to the plane exceeds α times the standard deviation from the mean distance to the plane are tagged as outlying. Hence a set $\dot{\mathcal{S}} = \{m \in 3, 4, \ldots N : 3 < \text{rank}(\ldots) < K + 1\}$ is defined such that

$$||\mathbf{P}_i - \mathbf{x}_q|| < \frac{1}{(K-2)} \sum_{m \in \dot{\mathcal{S}}} ||\mathbf{P}_i - \mathbf{x}_m||,$$

as depicted in Figure 5.12.

However, this proposed method failed to yield promising results, despite manually tuning the variable α. It could be argued that the assumption that the plane defined by a point and its two nearest neighbours approximates the manifold is wrong as can be seen in Figure 5.13. There could be a set of three points, all within the manifold, but in such proximity and relative orientation, that a plane through it, would not approximate the manifold, but rather cut through it, as can be seen in Figures 5.14 and 5.15.

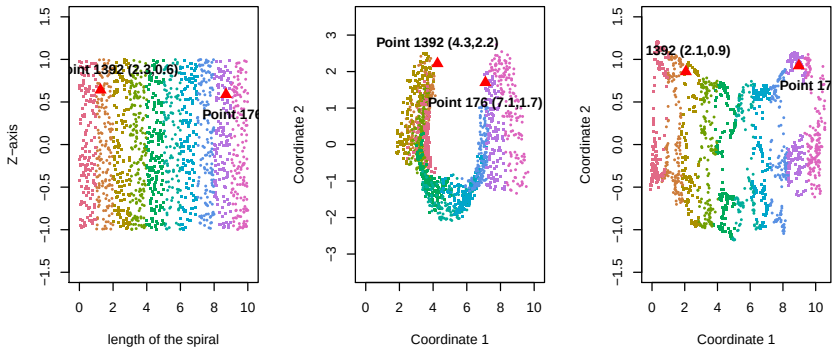


FIGURE 5.9: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Mean K-Nearest 1st and 2nd Level Neighbours with Limit

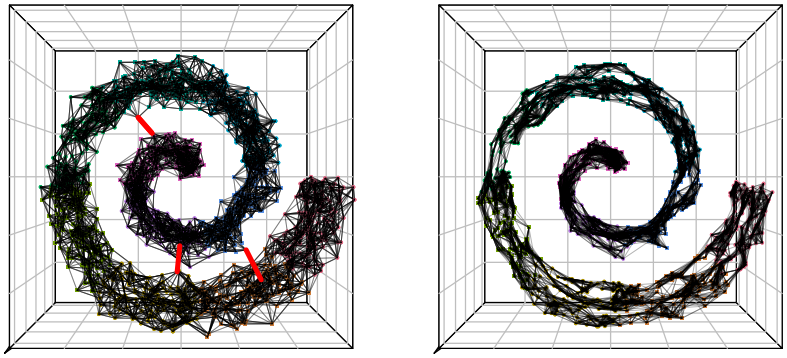


FIGURE 5.10: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section (Mean K-Nearest 1st and 2nd Level Neighbours with Dot Product Limit) variance (right)

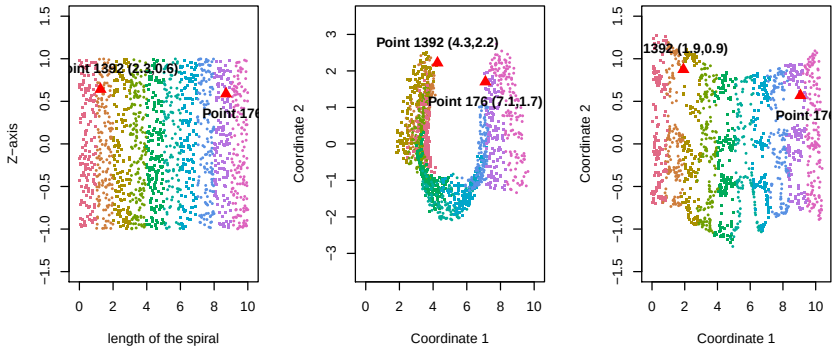


FIGURE 5.11: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Mean K-Nearest 1st and 2nd Level Neighbours with Dot Product Limit

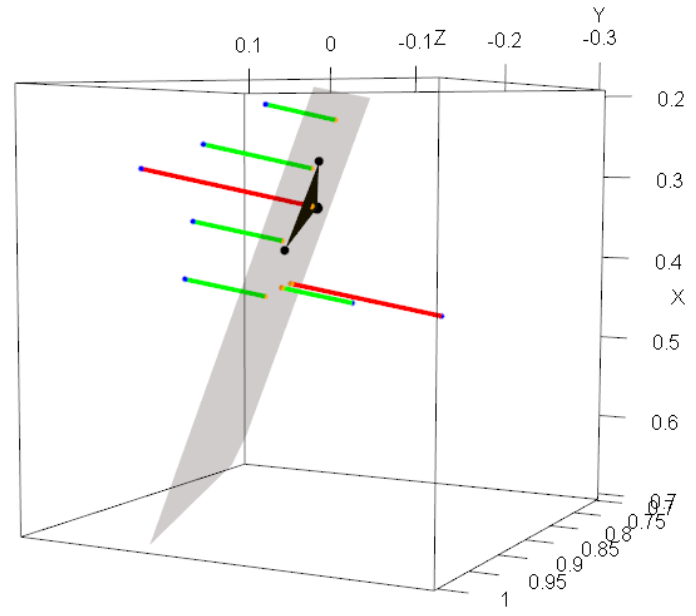


FIGURE 5.12: Plane through point **2** and its two nearest neighbours, with the other neighbours, showing distance to the plane (Section Distance to Plane).

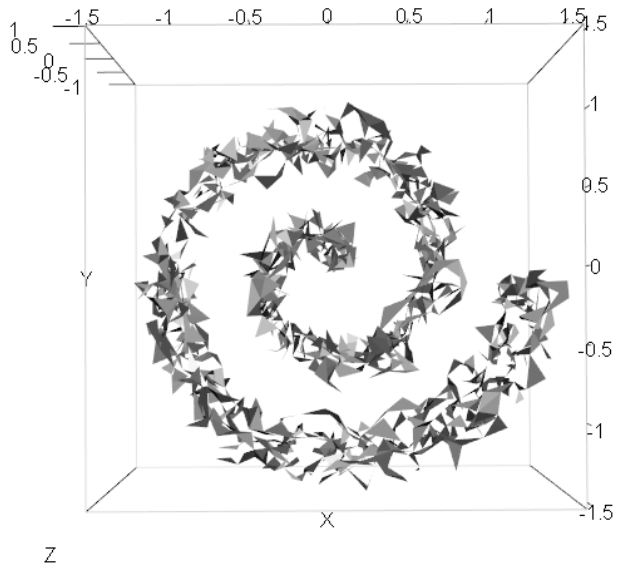


FIGURE 5.13: All triangular planes formed by each point and its two nearest neighbours (Section Distance to Plane).

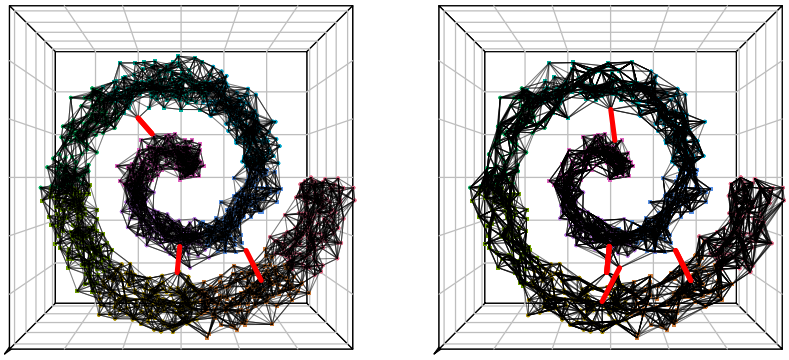


FIGURE 5.14: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section (Distance to Plane) variance (right)

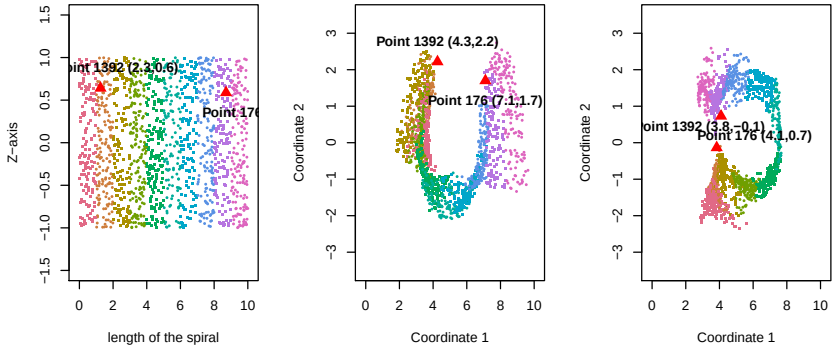


FIGURE 5.15: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Distance to Plane

5.7 Distance to Best Fit Plane

In this approach, it is assumed that the **plane** $\mathbf{P}_i$ defined as the best fit plane through each point $\mathbf{x}_i$ and all its nearest neighbours $\mathbf{x}_j$ to $\mathbf{x}_k$ approximates the manifold at point i. The best fit plane is achieved using PCA and the principle components with the largest eigenvalues (most variance explained). Each data point is then projected onto its best fit plane, as depicted in 5.16. The extend to which the plane approximates the manifold is dependent on the weight that the outliers have on all the points used in determining the best fit plane. However, increasing the number of points to decrease the effect of outliers, increases the risk of including points that are not local to that area of the manifold, thus resulting in poor manifold approximation, as can be seen in Figure 5.17. For illustrative purposes, Figure 5.18 depicts the relationship between the number of points used to fit the plane and the number of jumps between different areas of the manifold. This information about the nature of the manifold would probably not be available to the analyst in a real-world scenario, thus determining

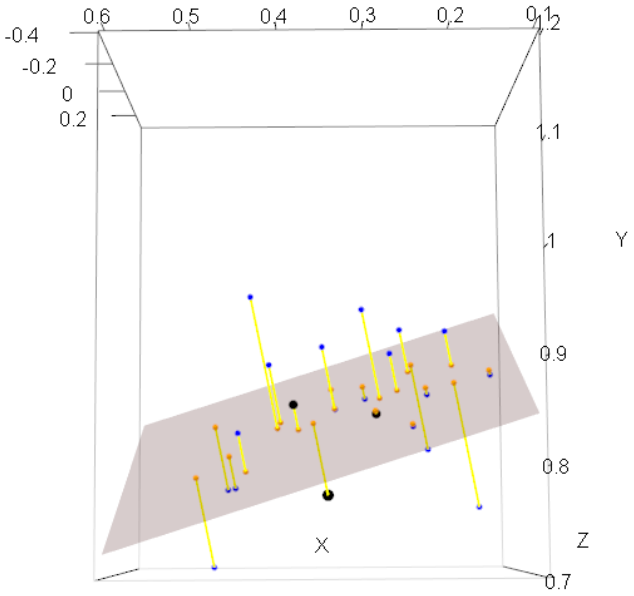


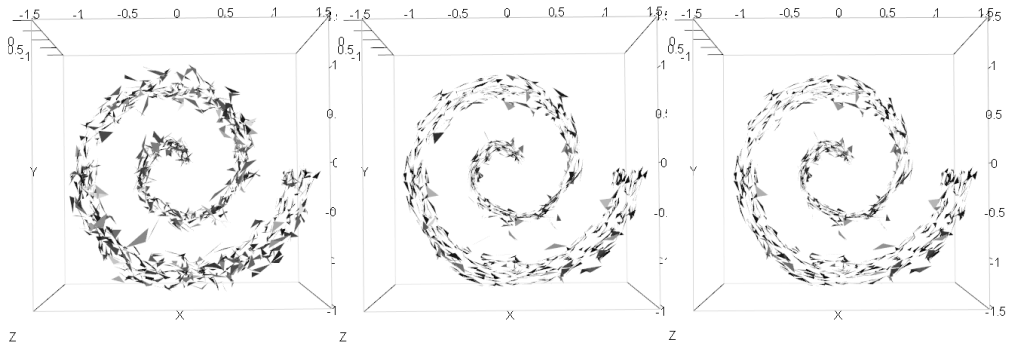
FIGURE 5.16: Best fit plane through point **2** and its 20 neighbours, showing distance to the plane (Section Distance to Best Fit Plane).

the optimal number of points to use cannot be predetermined in this fashion and will have to be refined through validation.

However, even with an optimised number of points to fit the plane, this proposed method, while promising, failed to remove all outliers. It could be argued that the assumption that the best fit plane approximates the manifold in all cases, is wrong as can be seen in Figure 5.17 panels A and C. There could be outliers in the neighbour set that carry such weight that they alter the orientation of the plane, regardless of the number of points in the subset, thereby resulting in jumps between different areas of the manifold (Figure 5.19) and poor MDS results (Figure 5.20).

5.8 Local Plane Angle

In this approach, as an enhancement of the previous approach, a best fit plane ($Pfit_i$) is fitted through each data point (i) and all its neighbours, as demonstrated in Figure 5.21. The angle between $Pfit_i$ and the X ($X = 0$), Y ($Y = 0$) and Z ($Z = 0$) planes are determined and used as reference angles (X_0, Y_0, Z_0). Subsequently, each neighbour is excluded one by one and a new plane ($Ptest_i$) is fitted over the remaining points and the angles between $Ptest_i$ and the X ,Y and Z planes are determined and used as test angles (X_{test}, Y_{test}, Z_{test}). Each point which, when excluded, causes the greatest difference between X_0 and X_{test}, Y_0 and Y_{test} and Z_0 and Z_{test}, is deemed an outlier and excluded from the neighbour set of point i. With these points excluded, a new plane $Pfin_i$ is fitted through point i and the remaining neighbours as a closer approximation



(A) 9 points (B) 20 points (C) 40 points

FIGURE 5.17: All triangular planes formed the projection of each point and its two nearest neighbours onto the best fit plane (Section Distance to Best Fit Plane) through each point and all its neighbours.

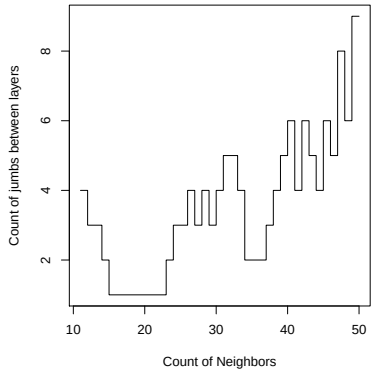


FIGURE 5.18: Showing the relationship between the number of points used to fit the plane (Section Distance to Best Fit Plane) and the number of jumps between layers of the Swiss-Roll.

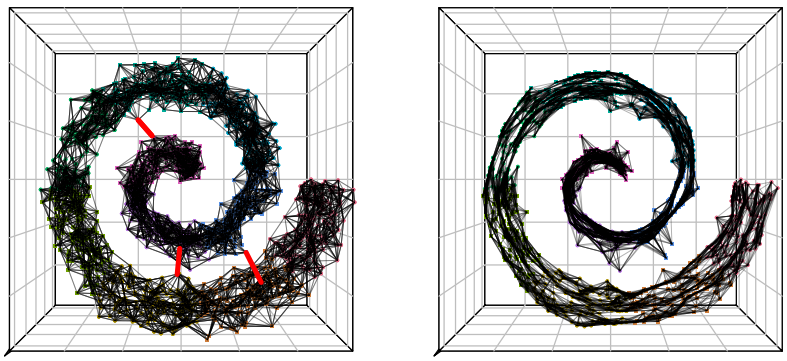


FIGURE 5.19: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section Distance to Best Fit Plane) variance (right)

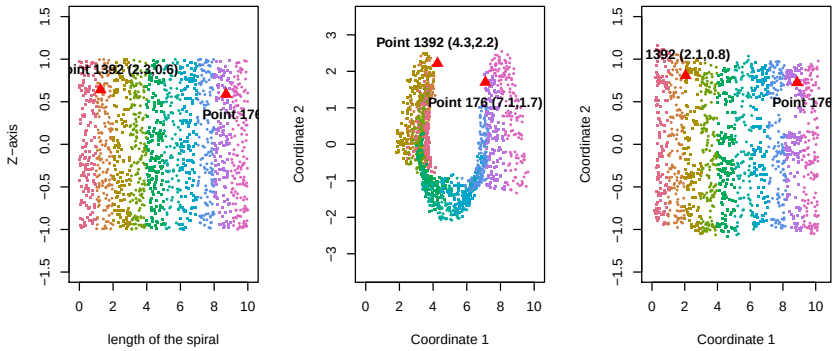


FIGURE 5.20: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Distance to Best Fit Plane

to the manifold. Each data point is then projected onto its revised plane and the distance graph determined from this altered dataset. Choosing the optimal number of neighbours to use in the fitting of the original plane $Pfit_i$ and also the number of points to retain for the revised fitted plane $Pfin_i$ is governed by the tradeoff between the sizable impact of outliers on a small neighbour set, versus the inclusion of non-local data points in a large neighbour set and should be optimised through validation as potential future work. Figure 5.22 depicts the relationship between the number of points used to fit the plane and the number of jumps between different areas of the manifold. The twenty nearest neighbours to each data point were selected and the remediation performed to achieve the smoothed out dataset, seen in Figure 5.23. The resultant smoothed Swiss-roll is visible in Figure 5.24 and the positive MDS representation is shown in Figure 5.25.

5.9 Shared Neighbours

Another approach is to count the number of neighbours N' which also has the original data point $\mathbf{x}_i$ as a neighbour. Each point is then classified as "point and neighbours in manifold" when less than 10% of the neighbours (determined through observations by reasercher) are not shared (common to both the data point and its neighbour), "original data point not in manifold" when less than half of the neighbours are not shared, while the rest of the points are classified as "some neighbours are not in the manifold". If **all** points are deemed in the manifold, nothing is changed, if **some** points are deemed not in the manifold, the **original** point is projected onto a best fit plane through point and its *shared* neighbours and if the **original** point is deemed not in the manifold, it is moved to its nearest *shared* neighbour. The number of nearest neighbours to consider can be adjusted through validation to find the optimal solution - see Figure 5.26 for the relationship between the number of neighbours and the jumps between areas on the manifold. Once remediation is applied to the dataset,

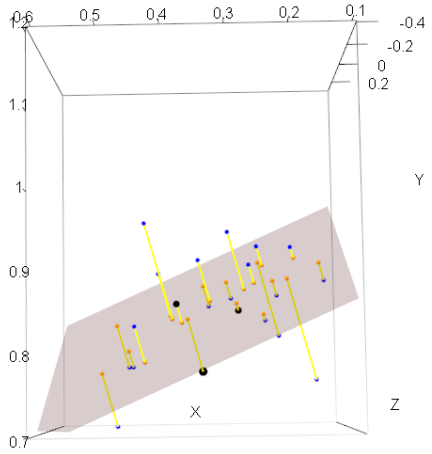


FIGURE 5.21: Best fit plane through point **2** and its 20 neighbours, showing distance to the plane (Section Local Plane Angle).

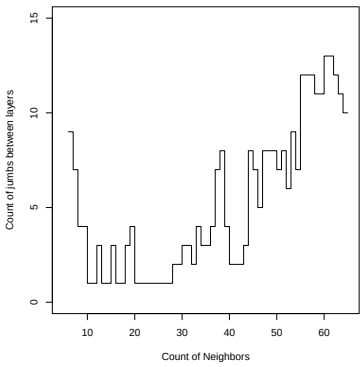


FIGURE 5.22: Showing the relationship between the number of points used to fit the plane (Section Local Plane Angle) and the number of jumps between layers of the Swiss-Roll.

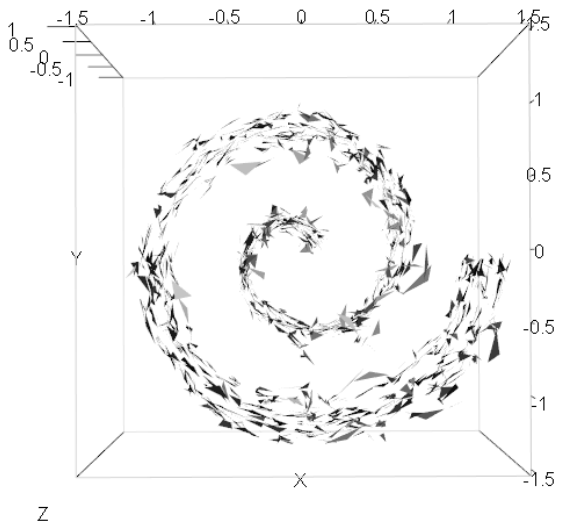


FIGURE 5.23: All triangular planes formed the projection of each point and its two nearest neighbours onto the best fit plane (Section Local Plane Angle) through each point and all its neighbours.

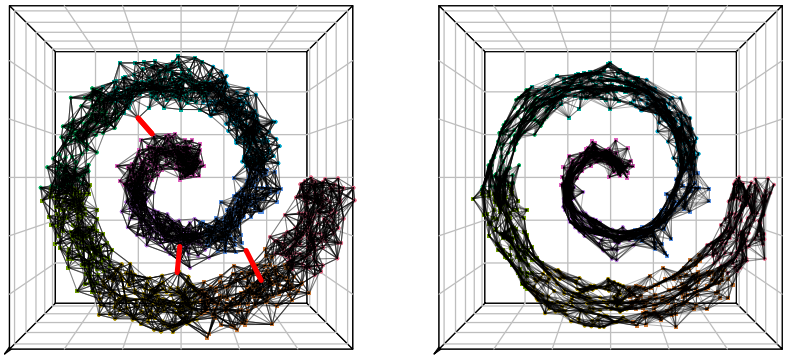


FIGURE 5.24: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (Section Local Plane Angle) variance (right)

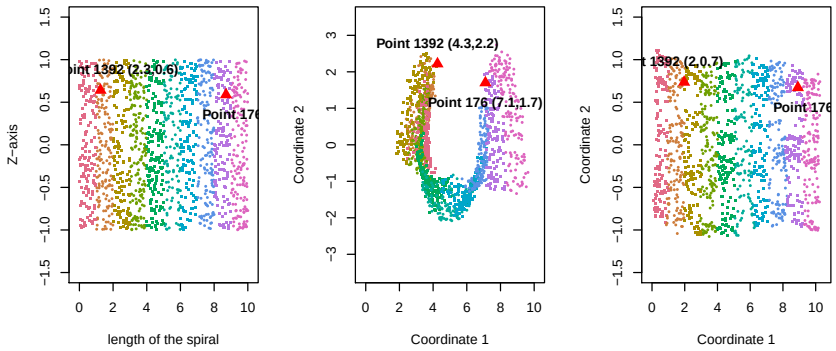


FIGURE 5.25: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Local Plane Angle

the original number of neighbours also needs to be adjusted to find paths between all data points. Choosing the count of neighbours as 20 produces a smoothed Swiss-roll as seen in Figure 5.27, resulting in good dimensionality reduction - Figure 5.28.

5.10 Cube Density

A final approach could be to divide the entire input space into equal cubes and to count the number of data points in each, with cubes containing only **one data point** labeled as "sparse". This process is repeated with the cube matrix moved in each dimension by **half the side-length of one cube**, thus producing a new set of cubes, some labeled as "sparse" in a similar fashion. Subsequently, all points that are common to both sets of **sparse** cubes are deemed outliers and moved to its nearest neighbour. It is possible to vary the size of the cube to optimise the result, as can be seen in Figure 5.29. This approach produces a smoothed Swiss-roll as seen in Figure 5.30, resulting in good dimensionality reduction - Figure 5.31.

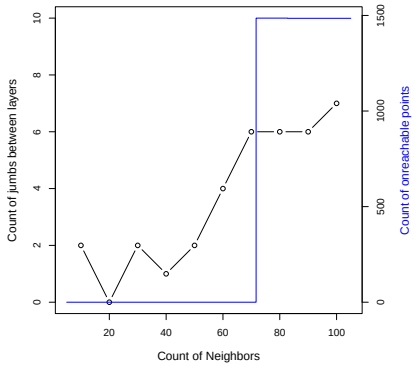


FIGURE 5.26: Showing the relationship between the number of points used to fit the plane (Section Shared Neighbours) and the number of jumps between layers of the Swiss-Roll.

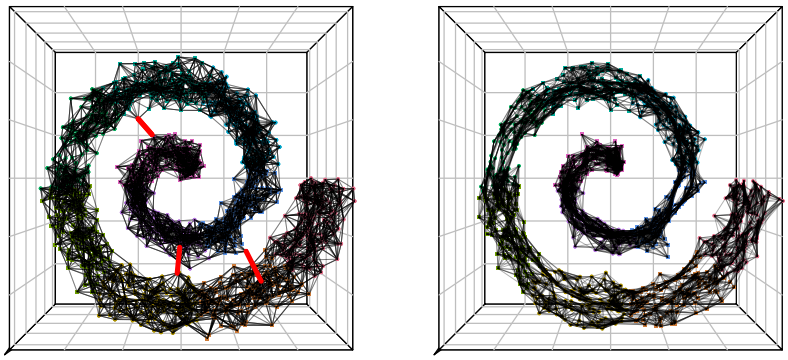


FIGURE 5.27: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (Section (Shared Neighbours) variance (right)

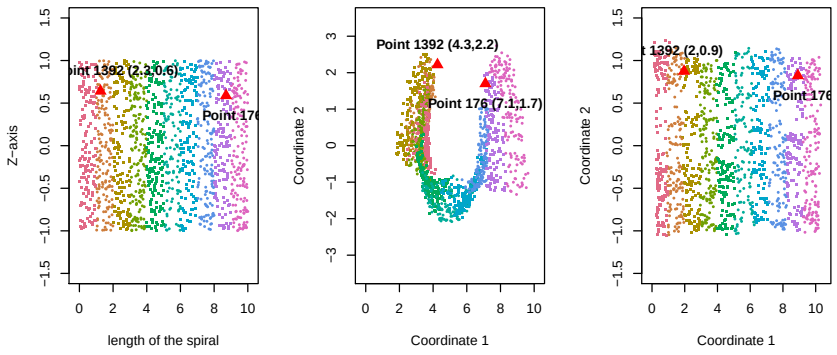


FIGURE 5.28: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Shared Neighbours

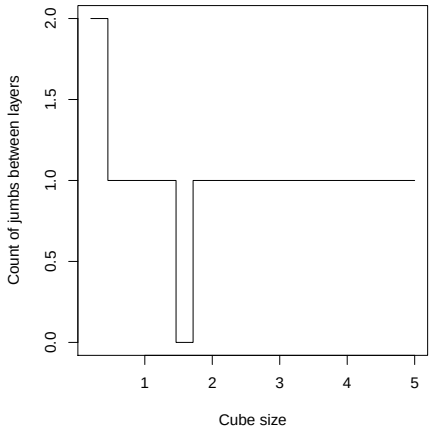


FIGURE 5.29: Showing the relationship between the size of the cube (Section (Cube Density) and the number of jumps between layers of the Swiss-Roll.

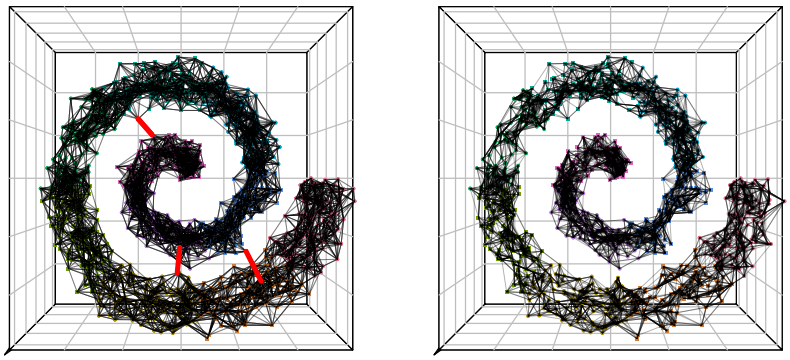


FIGURE 5.30: Swiss-Roll with orthogonal variance (left) vs. Swiss-Roll with remediated (according to Section (Cube Density) variance (right)

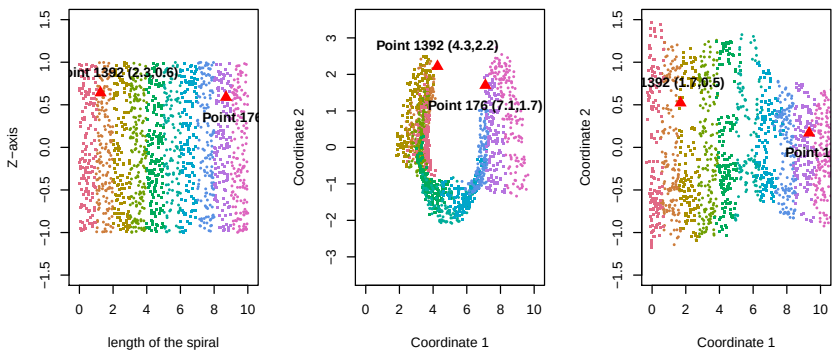


FIGURE 5.31: Flat surface rolled into Swiss-Roll (left) vs. 2-dimentional MDS representation of the Swiss-Roll with variance before (middle) and after (right) remediation according to Section Cube Density

To evaluate the performance of each of the remediation proposals, the root of the mean of the squares (RMS) (Kostina, Tzvetkov, and Serov [2020]) of the inter-point distances of each 2-dimensional output dataset is compared to the RMS of the distance matrix of the original flat surface which was used to construct the Swiss-roll. The **Distance RMS** is given by

$$\sqrt{\frac{1}{N}\sum_{i,j\in 1\ldots N}||\mathbf{d}_X(ij)||^2}$$

and the **Distance RMS ratio** is calculated as

$$1-\frac{|Distance\ RMS_{MDS\ surface}-Distance\ RMS_{flat\ surface}|}{Distance\ RMS_{flat\ surface}}$$

where **flat surface** is the original flat surface from which the Swiss-roll was created and **MDS surface** is the 2-dimensional plane formed by applying classical MDS to the distance graph of the Swiss-roll. The **Distance RMS ratio** can range from 0 indicating complete disalignment between the MDS surface and the original flat surface and a value of 1 would indicate a perfect replication of the original flat surface by MDS. Table 5.1 also provides the percentage of data points that were changed during the remedial process, where a lower number would indicate a less intrusive algorithm.

Even in the case of the Swiss-roll with no orthogonal variance, the distance RMS ratio is not 1 to 1 (0.9421 as seen in Table 5.1), which demonstrates that this procedure does not fully preserve the interpoint distances of the manifold, but is rather an approximation thereof. The use of linear Euclidean distances, not compensating for the local curvature of the manifold, could be a contributing factor; while small at individual point level, it could aggregate substantially over a large dataset. Table 5.1 ranks these RMS ratios, listing the Swiss-roll with **no** orthogonal variance in the 5^{th} position having a distance RMS ratio of 0.9421, the Swiss-roll **with** orthogonal variance in 11^{th} position (RMS ratio = 0.7725) and **Distance to Best Fit Plane** remediation ranking in the number one position with an RMS ratio of 0.979.

The efficacy of these remediations hinges on two factors: the ability to **detect** observations that hinder true manifold approximation, and how these problem points are **addressed**. As can be seen in the "Jumps" column in Table 5.1, the "Distance to plane" and "Cube Density" remediations did not effiectively identify **all** points that skew the manifold approximation, leading to low RMS ratios, which could point to poor performace. While these results apply to this particular experiment, it is expected to generalise to some degree, given that the methodologies applied, was in no way predicated on the Swiss-roll shape of the manifold, and yielded improvement over the case where no remediation was applied to the dataset with orthogonal variance in nine of the ten applications. In most cases, the detection of the problematic data points, and the manner in which to deal with it, is not linked, which opens up a multitude of remediation options which could be experimented with and the efficacy thereof tested through validation. Note here that the use of **RMS ratio** is not the be-all

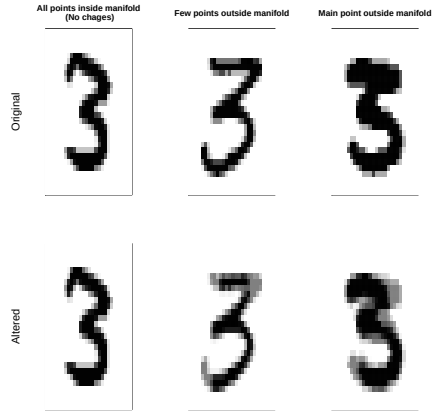


FIGURE 5.32: "Averaging" effect of Shared Neighbours remediation on the handwritten digit "3". More weight (darker greyscale) is placed on the the typical shape of the digit and less so on uncommon attributes, e.g. sharp bends and long tails are greyed out.

and end-all as the resulting representation could still be advantageous for purporses of prediction. Later in the project, some of these remediations are applied to cases where predictive performance is used for comparison.

To demonstrate the application and efficacy of these proposed remediations on a well known dataset, it is applied to the MNIST database of handwritten digits. A Boosted Logistic Regression model optimised for accuracy is trained and tested on various datasets. These datasets progress from the original dataset, the most relevant principle components of the original dataset, a dataset where ISOmap was applied or the original dataset and lastly the case where ISOmap was applied to the remediated dataset. The accuracy of the model against the test data and the percentage points improvement over the initial case, is listed in Table 5.2. The progressive improvement demonstrates the benefit of applying Tenenbaum's methodology and the additional benefit of preceding Tenenbaum's methodology with a remediation procedure to account for variability in the data, when resorting to dimensionality reduction in modeling of data.

Figure 5.32 shows the effect of remediation, in this case shared neighbours (Section 5.9) on the handwritten digits. It can be seen that the characteristics of the digits which make it look less like what is expected (the typical or "average" digit 3), e.g. the sharp bend in the top righthand corner of the middle 3, is softened (assigned less weight).

TABLE 5.1: Comparing Remediation proposals

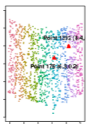
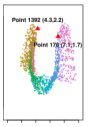
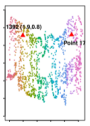
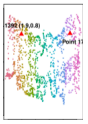
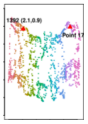
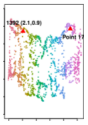
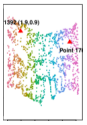
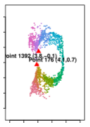
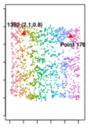
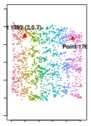
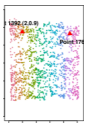
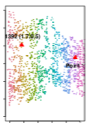
Section	Remediation	Points changed	Jumps	Distance RMS Ratio	Rank	Output
3	*No orthogonal variance in data*	*0*	*0*	*0.9421*	*5*	
4	*Variance with no remediation applied*	*0*	*7*	*0.7725*	*11*	
5.1	Mean of K-Nearest Neighbours	100%	0	0.9206	9	
5.2	Mean of K-Nearest Neighbours with Limit	56%	0	0.9610	4	
5.3	Mean K-Nearest 1st and 2nd Level Neighbours	100%	0	0.9361	7	
5.4	Mean K-Nearest 1st and 2nd Level Neighbours with Limit	8%	0	0.9340	8	
5.5	Mean K-Nearest 1st and 2nd Level Neighbours with Dot Product Limit	100%	0	0.9409	6	
5.6	Distance to Plane	27%	5	0.6593	12	
5.7	**Distance to Best Fit Plane**	**100%**	**0**	**0.9790**	**1**	
5.8	Local Plane Angle	100%	0	0.9700	2	
5.9	Neighbour of Neighbours	62%	0	0.9611	3	
5.10	Cube Density	21%	1	0.8785	10	

TABLE 5.2: Table showing the percentage points (ppts) improvement on accuracy when applying Tenenbaum et al. [2000] and remediation to the MNIST dataset

Dataset	**Train Accuracy**	**Test Accuracy**	**Train Improvement**	**Test Improvement**
Original dataset as is	91%	89%		
PCA on Original dataset	90%	88%		
ISOmap on Original dataset	92%	89%	+2 ppts	+1 ppt
ISOmap on Remediated dataset	95%	90%	+3 ppts	+1 ppt

6 Computational Gains in Path Selection

One of the key elements of the methodology proposed by Tenenbaum et al. [2000] is the finding of the shortest path between all points/vertices in the dataset - the so-called geodesic (approximant) distances on the manifold. For datasets with a small number or observations this is easily computed and yields reliable results. However, the main purpose of the procedure is dimensionality reduction, thus one would most likely be expecting to use this on datasets with a large number of observations and high dimensionality, which introduces high computational requirements. So, for instance, when using Floyd's algorithm (Floyd [1962]), to find the shortest paths, if it exists, the process relies on iterative cycling through the dataset, updating the paths on each cycle/pass until all paths are deemed to be the shortest, i.e. no updates performed in a cycle. This process can be computationally intensive, which begs the question: is finding the **shortest** path critical to the success of Tenenbaum's procedure? Fortunately, the *curse of dimensionality* is beneficial in some sense, which we could apply to this problem. The increase in sparsity of high dimensional data, brings about the tendency that the shortest path between data points is often not much shorter than any other reasonal path between those points. To test the hypothesis that an increase in dimensionality would decrease the discrepancy between **any** (reasonal) path and the shortest path between points in dataset, a **stochastic any-paths algorithm** was developed to determine the distance between vertices on a graph which relaxes the constraint that the path between two points on the graph be the **minimum distance** path.

Floyd's algorithm permutes over pairwise vertices, "connecting" paths, should any exist and be shorter than an existing path. One permutation over all points constitutes a pass over the distance matrix after which the distance matrix is thus updated reflecting the revised path distances. For a sufficient number of passes (N in the case of Floyd's algorithm), if such paths exist, they will be the shortest paths on the graph between all pairs of vertices. Floyd's algorithm thus presents a growth rate of $O(N^3)$.

In the stochastic any-paths algorithm, paths are connected probabilistically and passes are only made until the updated distance matrix is populated by finite path lengths. Therefore, as soon as all pair-wise points are connected, no more passes are made.

This still requires the same number of operations as Floyd's algorithm but, provided that paths exist, they will be found in **at worst** $O(N^3)$.

Herewith the stochastic any-paths algorithm: Let $\mathbf{D} = (d_{ij})_{N\times N}$ denote a matrix of distances/edge-lengths between vertices on a graph G. For points which are not connected as neighbours, distances are recorded as infinite.

Initialise: Define $\mathbf{D}^{(k)} = (d_{ij}^{(k)})_{N\times N}$ and set $\mathbf{D}^{(1)} = \mathbf{D}$. Set $k = 1$.

1. If all $d_{ij}^{(k)}$ are finite, return $\mathbf{D}^{(k)}$, otherwise:
   (a) For $i, j \in 1, \ldots, N$, calculate $\mathbf{p} = (p_r)_{N\times 1}$ where
   $$p_r = \begin{cases} 1/(d_{ir} + d_{rj}), & \text{for } i \neq j \neq r, \\ 0, & \text{otherwise.} \end{cases}$$
   with $r = 1, 2, \ldots, N$.
   (b) Select an index $r*$ from the set of integers $\{1, 2, \ldots, N\}$ with distribution p/p'[1] and set
   $$d_{ij}^{(k+1)} = min(d_{ij}^{(k)}, d_{ir*}^{(k)}, d_{r*j}^{(k)})$$
   .
2. Set $k \to k + 1$. Return to step 1.

For practical verification, both the shortest path, and any-paths algorithms where applied to an uniformly distributed unit radius spheriod dataset where data points are distributed inside the spheriod and not on the surface. A grid-search was performed where the number of dimensions of the spheriod was gradually increased, noting the ratio of the stochastic any path distance and the shortest path distance between all data point pairs. Figure 6.1 depicts three histograms of this any path to shortest path ratio for the spheriod with three, fifty and hundred dimensions. It can be seen that as the dimensionality increases, the majority of the interpoint distances tend to have a $1:1$ any path to shortest path ratio, supporting the hypothesis that for high dimensional data, the shortest path between data points is often not much shorter than any other reasonal path between those points. It is also noted that as the dimensionality increases, there is a grouping of distances with an any path to shortest path ratio of approximately 1.4. Closer inspection reveals that this coincides with twice the average interpoint distance of the dataset chosen for this analysis (spheriod). As seen in Figure 6.2, the interpoint distances cluster around multiples of half the average interpoint distance of the dataset (≈ 1.266 in the case of a spheriod with 100 dimensions, 350 data points and unit radius. This ratio of approximately 1.4 was achieved regardless of the number of data points, the dimensionality or the radius of the spheriod, and the origin thereof could not be confirmed.

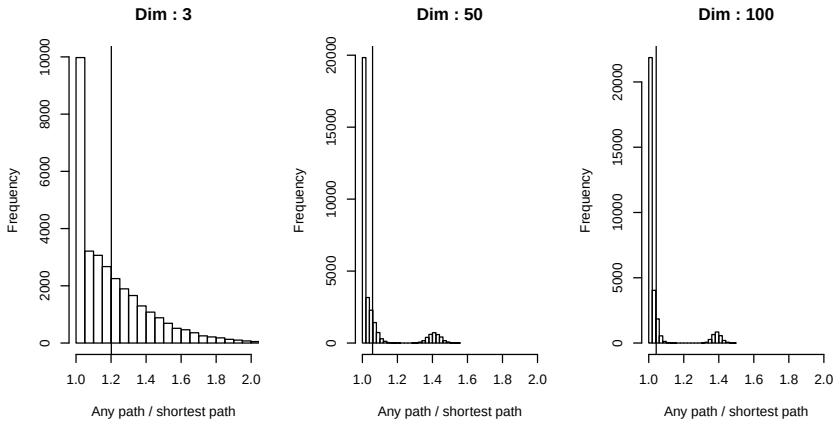


FIGURE 6.1: Histogram of the ratio of any path to shortest path distances for 3, 50 and 100 dimensions.

Figure 6.3 illustrates the 1:1 ratio of the number of **passes** to the number of **points** in the case of Floyd's shortest path algorithm (a.i) and a significantly lower ratio in the case of the stochastic any-paths algorithm (a.ii). The 1:1 ratio also holds true for the number of **steps** vs the number of **points cubed** of the shortest path algorithm (b.i), whereas, in the case of the stochastic any-paths algorithm (b.ii), again this ratio is much lower. It is also noted that while the performance of the shortest path algorithm remains constant relative to the number of points, the stochastic any-paths algorithm's performance increases with the number of points (lower passes to points ratio).

It should be noted that while there are other more numerically cost effective shortest path algorithms, e.g. A*(Hart, Nilsson, and Raphael [1968]), Dijkstra (Grama, Gupta, Karypis, and Kumar [2003]) and the rectangular algorithm Singh and Mishra [2014], the aim of this analysis was to prove that the shortest was not required and that such a constraint could be relaxed.

While being cognisant of the uncontrolled environment where this code was executed (the computer was not solely running this task) and the non-optimised R-based for-loop structure of the code, it is still interesting to see that the any-paths algorithm, for the simulation conducted here, is twice as fast as the shortest path algorithm, as can be seen in Figure 6.4

When applying the stochastic any-paths algorithm to the Swiss-roll dataset with **no** orthogonal variance, followed by classical MDS (as in previous chapters), the output is not rectangular as expected, but rather 'M'-shaped (like the McDonalds' M) - Figure 6.5**(d)**. However, along coordinate 1, the color bands are preserved, indicating that the predicted variable could still be extracted from this output.

Figure 6.6 illustrates that, as in the case of the shortest path algorithm, the stochastic any-paths algorithm fails to maintain the color bands along any of the MDS

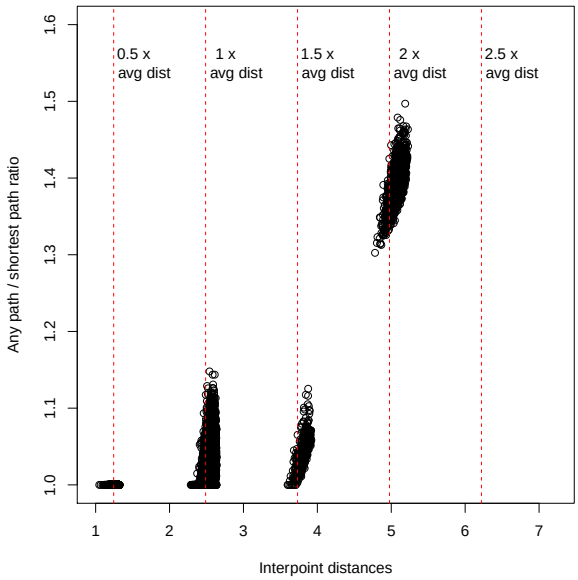


FIGURE 6.2: Scatterplot of the distances between points vs the any path to shortest path ratio (Dims = 100, N = 350).

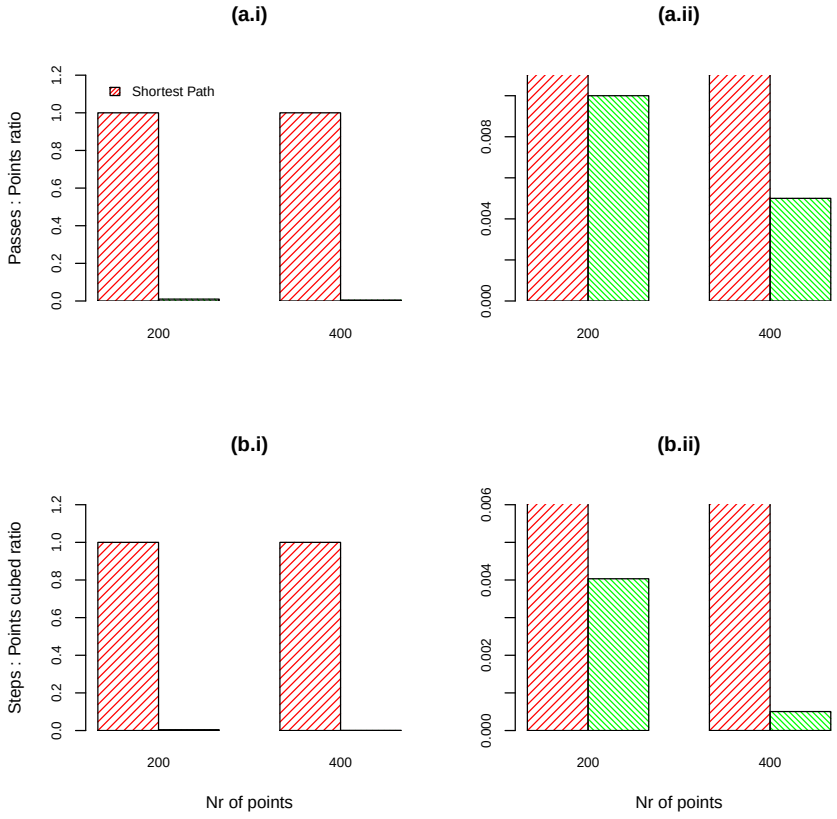


FIGURE 6.3: Comparing the number of passes and steps for shortest Path and any Path algorithm for different dataset sizes.
(a.i) Number of passes divided by number of points - full scale
(a.ii) Number of passes divided by number of points - y-axis zoom in
(b.i) Number of steps divided by number of points cubed - full scale
(b.ii) Number of steps divided by number of points cubed - y-axis zoom in

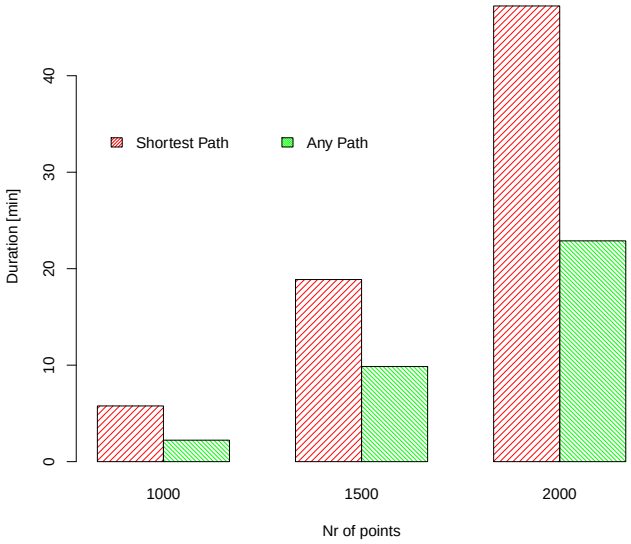


FIGURE 6.4: Comparing run duration for shortest Path and any Path algorithm for different dataset sizes.

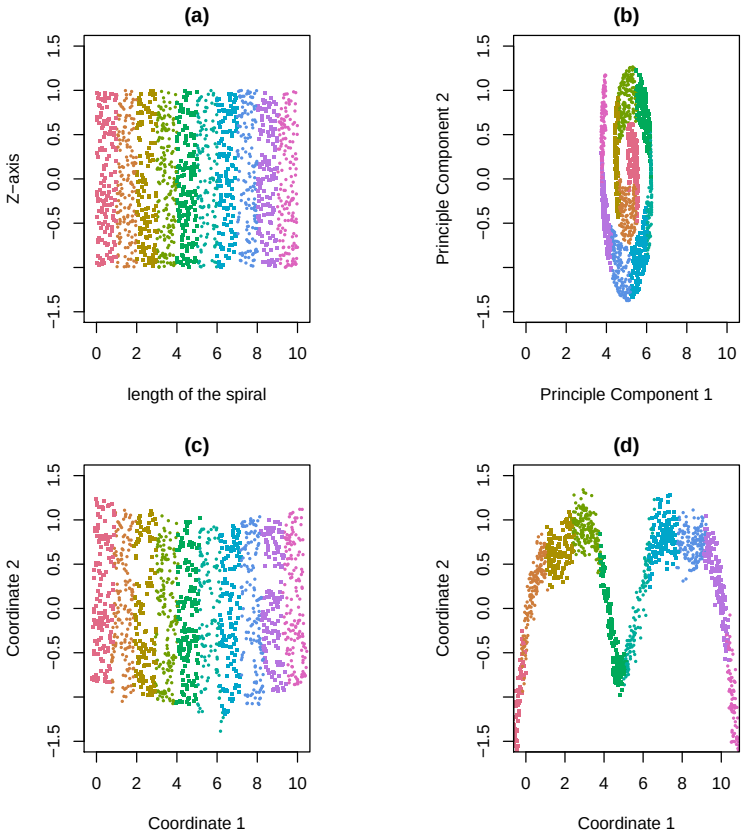


FIGURE 6.5: Comparing shortest Path to any Path output for Swiss-roll with **no** orthogonal variance. **(a)** Original flat surface, **(b)** First two principle components, **(c)** shortest Path algorithm and **(d)** any Path algorithm.

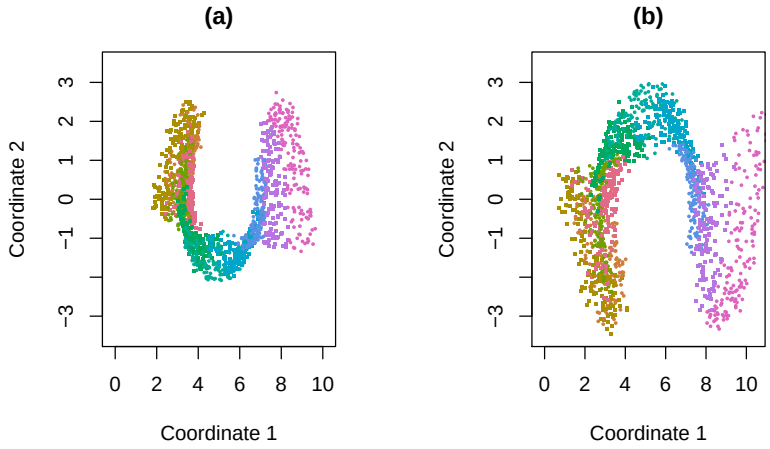


FIGURE 6.6: Comparing shortest Path to any Path output for Swiss-roll **with** orthogonal variance. **(a)** shortest Path algorithm and **(b)** any Path algorithm.

coordinates when applied to the Swiss-roll dataset **with** orthogonal variance, thus necessitating remediation.

As can be seen in Figures 6.7 and 6.8, when remediation is applied to the Swiss-roll dataset with orthogonal variance, MDS in the case of the stochastic any-paths algorithm returns a wavelike structure, but again, the color bands are preserved along coordinate 1, indicating predictability in the output. In Figure 6.7 *KNN Mean with Limit* remediation (Section 5.2) was applied, while in Figure 6.8, *Shared Neighbours* remediation (Section 5.9) was applied.

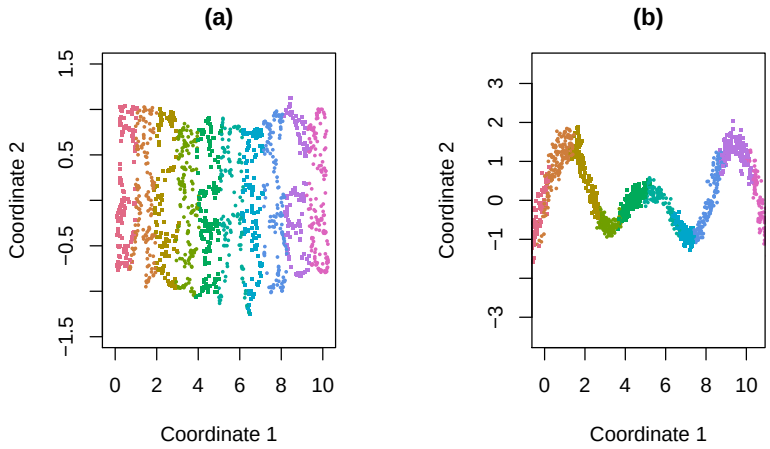


FIGURE 6.7: Comparing shortest Path to any Path output with **KNN Mean with Limit** remediation for Swiss-roll with orthogonal variance. **(a)** shortest Path algorithm and **(b)** any Path algorithm.

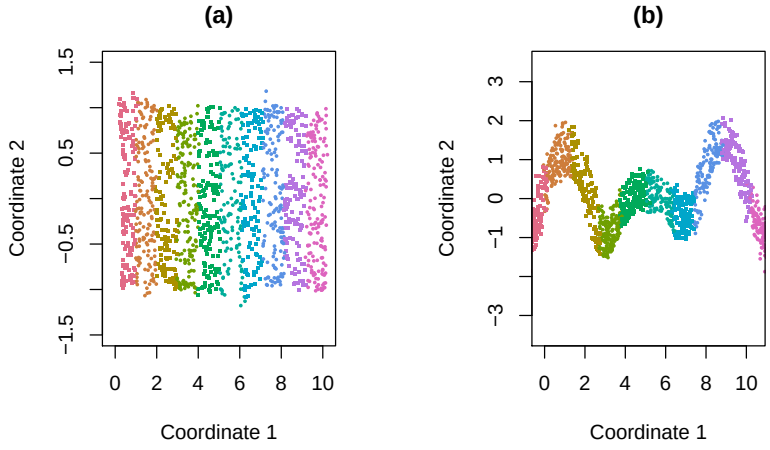


FIGURE 6.8: Comparing shortest Path to any Path output with **Neighbours of Neighbours remediation** for Swiss-roll with orthogonal variance. **(a)** shortest Path algorithm and **(b)** any Path algorithm.

7 Application of Proposed Remediations/Variations on Dimensionality Reduction on Real-world Data Sets

Having shown promising results on the Swiss-Roll and MNIST data, we now explore the application of the remediation techniques and the stochastic any-path algorithm on real-world data. It should be noted that the aim is to evaluate the performance uplift of the pre-processing and not the models. So, a simple model would suffice. Indeed, more complex models would only serve to make the contribution to performance uplift more opaque. Three datasets from KAGGLE (Goldbloom) were chosen for having numerical variables, more than 50 dimensions, not to be so large as to cause computational complexities and to be a classification problem for simpler comparative analysis. These are standard datasets used to test and assess methods in machine learning. Familiarity of the machine learning world with these datasets provide contextual insight to those readers, but the reader should adjust expectations accordingly as one cannot difinitively infer general applicability beyond the scope of the present results, but the features of the present analysis should be useful to guide expectations with respect to datasets with similar attributes.

The **first dataset** (referred to as the "**GESTURE**" dataset going forward) contains Electromyography (EMG) readings of muscle activity from eight sensors on a users forearm. These readings could potentially be used in a prosthetic control system to activate movement in a robotic prothesis such as open/close hand or rotate wrist. Readings for four hand gestures, i.e. "Rock", "Paper", "Scissors" and "OK" were recorded, generating a classification dataset with four classes. This dataset had no unused readings (columns) and none of the features had zero variance. There were quite a number of outliers that were discarded (using 6 times the IQR as a lower boundry and 7.5 times the IQR as an upper boundry 2238(20%) of the observations were discarded), resulting in a dataset of 9439 observations, 64 predictor variables (there were eight readings from each of the eight sensors) and one predicted variable with four classes (the gesture). This dataset was sampled and split into a training set of 1000 observations and a test set of 1000 observations. Having an equal amount

of training and testing observations is not a requirement of this procedure, but it is chosen as such to simplify calculations.

As a **pre-baseline** analysis, a Support Vector Machine (SVM) model with Radial kernel was developed from the training dataset and applied to the test dataset. A training accuracy of 99% and a test accuracy of 91% was achieved, but because the aim of the algorithm is dimensionality reduction, this is not regarded as the true baseline against which the algorithm should be compared.

The **first baseline** was developed by applying principle component analysis (**PCA**) training dataset and selecting the top 40 principle components (kept constant throughout the analysis), which explains 88% of the variance. A SVM model is then developed from this PCA training dataset and predictions performed on the test dataset, yielding training accuracy of 98% and test accuracy of 91%. Despite the drop in explained variance, the accuracy has remained constant. These values now form the first baseline.

To obtain the **second baseline**, PCA is preceded with manifold detection using the **ISOmap algorithm** as proposed by Tenenbaum et al. [2000]. When using personally developed code, it became evident that the underlying manifold, if one exits, is not simple to detect, possibly due to dominating stochastic components. Very high neighbour counts were required, and some unreachable data points had to be removed. Given these complexities, a training accuracy of 88% and a test accuracy of 70% were achieved, showing significant deterioration from the first baseline.

The **first test case** was achieved by preceding the ISOmap algorithm with "Mean of K-Nearest-Neighbours with Limit" **remediation** as described in Section 5.2. The tendancy of this remediation to cause clustering of observations, necessitated the removal of even more unreachable observations. A training accuracy of 79% and a test accuracy of 49% were achieved, showing a drastic reduction in test accuracy over the second baseline.

In the **second test case**, the shortest path algorithm in ISOmap was replaced by the **any-path algorithm** as described in Chapter 6. In contrast to the second baseline, the number of neighbours required to find paths between all observations reduced to expected values ($<$ 10% of observation count). This test case yielded a further reduction in accuracy with a training accuracy of 84% and test accuracy of a mere 30%. This demonstrates that using the any-path algorithm has significantly deteriorated the manifold detection process and subsequent modeling on this dataset.

The **second dataset** (referred to as the "**CHURN**" dataset going forward) contains anonymised customer data pertaining to their activity on a telecoms network over a period of three months separately. The activity includes various types of call and data volumes, (e.g. incoming calls, outgoing calls, local calls, data consumption), recharges, ARPU etc. The predicted variable is customer churn and is derived by inspecting the customer activity in the third month, i.e. should the customer not make or receive any calls and not upload or download any data, it is presumed that

TABLE 7.1: Table showing the train and test accuracy when applying ISOmap, remediation and any-path algorithm to the **GESTURE** dataset

Dataset	Train Accuracy	Test Accuracy	Train Improvement	Test Improvement
Original dataset	99%	91%		
PCA on Original dataset (1^{st} baseline)	98%	91%		
PCA on Geodesic distances (2^{nd} baseline)	88%	70%	-10 ppts	-21 ppts
PCA with Geodesic distances of Remediated dataset	79%	49%	-9 ppts	-21 ppts
PCA with Geodesic distances of Remediated dataset and Any Path algorithm	84%	30%	-5 ppts	-40 ppts

TABLE 7.2: Table showing the train and test accuracy when applying ISOmap, remediation and any-path algorithm to the **CHURN** dataset - SVM Radial Kernel

Dataset	Train Accuracy	Test Accuracy	Train Improvement	Test Improvement
Original dataset	88%	90%		
PCA on Original dataset	88%	90%		
PCA on Geodesic distances (baseline)	87%	90%	-1 ppt	0 ppts
PCA with Geodesic distances of Remediated dataset	87%	90%	0 ppts	0 ppts
PCA with Geodesic distances of Remediated dataset and Any Path algorithm	87%	90%	0 ppts	0 ppts

the customer has churned to another service provider. The dataset has 226 columns and 99999 rows from which irrelevant fields (e.g. actual dates and mobile numbers) were removed, columns with zero variance were removed and the churn flag added resulting in a final dataset of 179 predicator variables and one predicted variable, spanning 99999 observations. From this, a training dataset of 2000 observations and a test dataset of 300 observations were uniformly sampled without replacement (no bootstrapping).

An SVM with a Radial kernel (with Signa and C parameterisation) model on the dataset yielded train and test accuracy values of 88% and 90% respectively. Then, as was the case in the GESTURE dataset, the **first baseline** was developed by applying PCA (50 components explaining 96% of the variation) prior to modeling on the dataset. This yielded comparable results of 88% and 90% train and test accuracies.

As a second baseline example, ISOmap was applied, but unlike in the case of the GESTURE dataset, the number of neighbours required to avoid unreachable points were quite low (25). The results where almost unchanged from the first baseline with 87% and 90% as test and train accuracies. Applying remediation did not improve the accuracy, yielding train accuracy of 88% and test accuracy of 90%. Lastly, using the any-path algorithm did not improve or deteriorate the accuracy with train and test values of 89% and 90% respectively. Applying the remediation and using the any-path algorithm had less than 0.5% effect on the accuracy of the model.

The minimal impact of the ISOmap procedure on the accuracy of the SVM model with Radial kernel could indicate that the underlying structure of the data is **linear**. A SVM model with linear kernel was also fitted to this dataset and very similar results were achieved. It stands to reason that if a radial kernel does not significantly outperform

TABLE 7.3: Table showing the train and test accuracy when applying ISOmap, remediation and any-path algorithm to the **CHURN** dataset - SVM Linear Kernel

Dataset	Train Accuracy	Test Accuracy	Train Improvement	Test Improvement
Original dataset	87%	90%		
PCA on Original dataset	87%	90%		
PCA on Geodesic distances (baseline)	87%	90%	0 ppts	0 ppts
PCA with Geodesic distances of Remediated dataset	87%	90%	0 ppts	0 ppts
PCA with Geodesic distances of Remediated dataset and Any Path algorithm	87%	90%	0 ppts	0 ppts

TABLE 7.4: Table showing the train and test accuracy when applying ISOmap, remediation and any-path algorithm to the **CANCER** dataset

Dataset	Train Accuracy	Test Accuracy	Train Improvement	Test Improvement
Original dataset	91%	87%		
PCA on Original dataset	91%	88%		
PCA on Geodesic distances (baseline)	76%	55%	-15 ppts	-33 ppts
PCA with Geodesic distances of Remediated dataset	71%	56%	-5 ppts	+1 ppt
PCA with Geodesic distances of Remediated dataset and Any Path algorithm	68%	56%	-8 ppts	+1 ppt

the linear kernel, the application of ISOmap would not have a meaningful impact, as can be seen when comparing the results for the linear kernel in Table 7.3 to the radial kernel results in Table 7.2

The **third** (and last) **dataset** (referred to as the "**CANCER**" dataset going forward) contains physical features of cell nuclei as obtained from digitised images produced by a fine needle aspirate (FNA) of a breast mass. This dataset has 565 observations, thirty predicator variables and one predicted variable indicating the malignant or benign state of the cell. Given that this is a simple model, an 80/20 train/test split is performed on the dataset as it provides a good balance between training and validation. Validation always has an optimistic bias so it requires more observations, provided that enough training samples remain to get a good fit. When linear kernel SVM is applied, the model yielded 91% and 87% train and test accuracy respectively. Applying PCA (25 components with 97% variance explained) prior to modeling, did not affect the accuracy much, yielding 91% and 88% for train and test accuracy respectively.

When applying ISOmap to the dataset, unlike the GESTURE dataset, a relatively low neighbour count (17%) was selected with successful path generation. It yielded similar training, but somewhat degraded test accuracies of 71% and 56% respectively. When applying remediation prior to modeling, both the training and test accuracy remained fairly constant at 68% and 68% respectively. The training accuracy deteriorated further when the any-path algorithm was used in stead of the shortest path algorithm yielding 68% for train and 65% for test accuracy.

From this analysis we found that applying remediation has decreased or maintained but not increased accuracy levels, indicating that the application of such remediation is not guaranteed to improve the modeling, and could lead to loss of accuracy. The

need for the remediation as proposed in this paper stems from the presence of stochastic elements masking the true manifold. It is thus only through successful remediation, should it exist, that the manifold is revealed. However, it would be a topic for further research to investigate the methods on how to pre-determine the presence of a manifold.

The use of the any-path algorithm to replace the shortest path algorithm did not impact the accuracy of the models in all three datasets. This indicates that if the need for a more numerically cost effective algorithm is required, the stochastic any-path algorithm can indeed be considered.

8 Conclusions

The unprecedented increase in the number of observations that has changed the statistical landscape and also the shape of data, has lead to many advances in the theory and implementation of non-linear dimensionality reduction over the last 20 years. In this paper we research and experiment with various remedial methodologies to improve on the manifold detection put forward by Tenenbaum et al. [2000]. From the ten remedial proposals, "Distance to Best Fit Plane" as described in Section 5.2 shows the most promising results. If the researcher/data scientist has *a priori* reasons to believe that any of the potential pitfalls of dimensionality reduction will be obsereved, the proposed remedies should be tried. Indeed, in cases where it is not obvious that such pitfalls could be observed, such analyses could reveal it to the researcher/data scientist. In most cases, the detection of the problematic data points, and the manner in which to deal with it, is not linked, which opens up a multitude of remediation options which could be experimented with and the efficacy thereof tested through validation.

While most of the remedies show an improvement of interpoint distance replication in dimensionality reduction, the true test lies in the improvement of the predictive nature of the data. Marginal predictive improvement is shown in the case of the MNIST dataset, but when applied to three other real world datasets (Chapter 7), the results show equal predictive accuracy at best, and in two of the cases show a reduction thereof. The first potential reason for this can be the lack of non-linear attributes, for which the researcher/data scientist can test and compare the performance of linear and non-linear methodologies to gain more insights into the underlying data structure. It could be a topic of further research to investigate the methods on how to pre-determine the presence of a manifold, and the nature thereof, be it linear or not. This paper also excluded manifolds with essential loops, e.g. the torus, and nonconvex manifolds, which could be explored through further study. Secondly, the presence of stochastic elements could also lead to the unsuccessful "revealing" of the non-linear manifold. It is through experimentation and validation of the various remedial methodologies proposed, that the influence of the stochastic elements could be minimisted in order to detect the non-linear manifold. Many of these methodologies have tuneable parameters which could be optimised through grid-searches. It should be noted that some of the remedies proposed, have an impact on the data (alters the datapoints as seen in the "Points changed" column in Table 5.1), and as such, the data scientist should always be mindful of the extend to which

the remedy impacts the data. This should not only be measured by the number of points changed, but also to the extend of each change. Future research could reveal tests which would assist the data scientist to choose the best remedial action with the smallest impact on the original dataset, and possibly provide optimal values for the tuneable parameters.

Chapter 6 investigates the possible use of the any-path algorithm for cases where there is a need for a more numerically cost effective algorithm. It was found that the any-path algorithm did not impact the accuracy of the models tested in this paper, indicating that it can be considered, should the need arise. Future research could reveal the conditions under which this algorithm should and should not be considered, and possibly also provide methods to estimate the benefits in run-time and computational resources.

A Appendix A - R Code for Swiss-Roll Generation

```
# -- CONSTANTS ----------------
N = 1500                    # Number of points
Turns = 2                   # Number of turns in the spiral
R_begin = 0.1               # Stating radius of the spiral
R_step = 3                  # delta R in 1 turn
Z_max = 5                   # Z - width of the swissroll
Z_start  = -Z_max/2         # Z - start of the swissroll
Xf_max = 10                 # Xf - lengh of the swissroll
Yf_max = Z_max              # Yf - width of the swissroll
Yf_start  = -Yf_max/2       # Yf  - start of the swissroll
Xf_bw = 1                   # width of the bands
palette(rainbow(Xf_max))
# -- SPIRAL FUCNTIONS ---------
s_preint = function(hoek,a){return(sqrt((R_begin + a*hoek)^2 + a^2))}
s_spiral = function(hoek,a){return(integrate(s_preint,lower = 0, upper =
    hoek, a = a)$value)}
f_hoek = function(hoek, s_hoek, a){return(abs(s_spiral(hoek, a) - s_hoek))
    }
f_Aconst = function(a, hoek, s_hoek){return(abs(s_spiral(hoek, a) - s_hoek
    ))}
# -- FLAT surface -------------
Xf_unif = floor(runif(N,min = 1,max = Xf_max+0.9999999))-0.5
Xf_band = runif(n = N, min = -Xf_bw/2, max = Xf_bw/2)
Xf = Xf_unif + Xf_band
P1 = which(Xf==min(Xf))
Yf = runif(N,min = Yf_start,max = Yf_start+Yf_max)
# -- Archimedean spiral ---------
# Archimedean spiral
Aconst = optim(par = c(R_step), fn = f_Aconst, s_hoek=Xf_max, hoek = Turns
    *2*pi, method = c("L-BFGS-B"))$par
R_step = Aconst*2*pi
Aamd = seq(from = 0, to = (Turns*2*pi), length.out = N)
Ramd = R_begin + Aconst * Aamd
Xamd = Ramd*cos(Aamd)
Yamd = Ramd*sin(Aamd)
Zamd = Yf
XYZamd = cbind(Xamd,Yamd,Zamd)
# -- FLAT surface rolled up ------
Aa = vector(length = N)
```

```
for(sh in c(1:N)){Aa[sh] = optim(par = c(Aamd[sh]), fn = f_hoek, s_hoek=Xf
    [sh], a = Aconst, method = c("L-BFGS-B"))$par}
R = R_begin + (Aconst * Aa)
X = R*cos(Aa)
Y = R*sin(Aa)
Z = Yf
XYZ = cbind(X,Y,Z)
# -- PLOT ------------------------
par(mfrow = c(1, 2), mar = c(5, 4, 5, 4))
# c(bottom, left, top, right)
plot(Xf,Yf,
     type = "p",
     pch = 20-(5*((Xf_unif+0.5)%%2)),
     cex = 0.4,
     col = Xf_unif+0.5,
     xlim = c(-0.2,0.2+Xf_max),
     ylim = c(Yf_start-0.5,0.5+(Yf_start+Yf_max)),
     xlab="Lenght of the spiral", ylab="Z-axis",
     main=paste("Flat surface with bands of normally distributed points","
         to be rolled into a Swiss-Roll",sep = '\n'))
plot(X,Y,
     type = "p",
     pch = 20-(5*((Xf_unif+0.5)%%2)),
     cex = 0.4,
     col = Xf_unif+0.5,
     xlab="X-axis", ylab="Y-axis",
     main=paste("Flat surface rolled into Swiss-Roll","viewed along Z-axis
         ",sep = '\n'))
abline(h=0,v=0,col = 'grey')
# -- END ------------------------
```

B Appendix B - R Code for Remediation on Swiss-Roll with varaince

```
Remediations = function(datapoints, NN = 2, method = 'local_mds', alfa =
    1.3){
  NNN = NN*1
  dp = datapoints
  ddp = dim(datapoints)[1]
  ddpts = dim(datapoints)[1]
  XYZ_new = dp      #matrix(NA,ddp,3)
  print(paste0(method," implementing..."))
# ----------------------------------------
  if(method == 'KNN_mean'){
    dpn = KNNF(dp,NN,returnall = FALSE)
    for(ii in c(1:ddp)){
      XYZ_new[ii,] = cbind(mean(dp[dpn$neighbours[ii,],1]),mean(dp[dpn$
          neighbours[ii,],2]),mean(dp[dpn$neighbours[ii,],3]))
    }
  }
# ----------------------------------------
  else if(method == 'KNN_mean_lim'){
    dpn = KNNF(dp,NN,returnall = FALSE)
    ptch = 0
    for(ii in c(1:N)){
      mn_edge = mean(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      sd_edge = sd(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      for(jj in c(1:NN)){
        nnch = 0
          if(dpn$edges[ii,jj] > alfa * mn_edge){
            dpn$neighbours[ii,jj] = NA
            nnch = nnch + 1
          }
      }
        XYZ_new[ii,] = cbind(mean(dp[dpn$neighbours[ii,],1],na.rm = TRUE),
                             mean(dp[dpn$neighbours[ii,],2],na.rm = TRUE),
                             mean(dp[dpn$neighbours[ii,],3],na.rm = TRUE))
        if(nnch > 0){
          ptch = ptch + 1
        }
    }
```

```
    print(paste0("nr of points changed = ",ptch))
  }
# ---------------------------------------
  else if(method == 'KNN_mean_2nd_nhbr'){
    dpn = KNNF(dp,NN,returnall = FALSE)

    for(ii in c(1:ddp)){
      mknnd = matrix(NA,NN+1,NN)
      mknnd[1,] = dpn$neighbours[ii,]
      for(jj in c(1:NN)){
        mknnd[(jj+1),] = dpn$neighbours[dpn$neighbours[ii,jj],]
      }
      XYZ_new[ii,] = cbind(mean(dp[as.vector(mknnd),1]),mean(dp[as.vector(
          mknnd),2]),mean(dp[as.vector(mknnd),3]))
    }
  }
# ---------------------------------------
  else if(method == 'KNN_mean_2nd_nhbr_lim'){
    dpn = KNNF(dp,NN,returnall = FALSE)
     ptch = 0
    for(ii in c(1:ddpts)){
      mknnd = matrix(NA,NN+1,NN)
      mknndd = matrix(NA,NN+1,NN)
      mknnd[1,] = dpn$neighbours[ii,]
      mknndd[1,] = dpn$edges[ii,]
      for(jj in c(1:NN)){
        nnch = 0
        mknnd[(jj+1),] = dpn$neighbours[dpn$neighbours[ii,jj],]
        mknndd[(jj+1),] = dpn$edges[dpn$neighbours[ii,jj],]
        if(mean(mknndd[(jj+1),]) > alfa * mean(mknndd[1,])){
          mknnd[(jj+1),] = NA
          nnch = nnch + 1
        } else {
            mknnd[(jj+1),] = dpn$neighbours[dpn$neighbours[ii,jj],]
        }
      }
      XYZ_new[ii,] = cbind(mean(dp[as.vector(mknnd),1],na.rm = TRUE),
                           mean(dp[as.vector(mknnd),2],na.rm = TRUE),
                           mean(dp[as.vector(mknnd),3],na.rm = TRUE))
      if(nnch > 0){
          ptch = ptch + 1
        }
    }
     print(paste0("nr of points changed = ",ptch))
  }
# ---------------------------------------
  else if(method == 'KNN_mean_dot'){
      dpnn = KNNF(dp,NNN,returnall = FALSE)
       ptch = 0
      for(ii in c(1:ddpts)){
      ndot = matrix(NA,4,length(combn(NNN+1,2,simplify = TRUE)))
      ndi = 1
```

```
87        for(jj in c(2:(NNN-1))){
88          nnch = 0
89          for(kk in c((jj+1):NNN)){
90            ndot[1,ndi] = ii
91            ndot[2,ndi] = jj
92            ndot[3,ndi] = kk
93            ndot[4,ndi] = abs(dot(dp[dpnn$neighbours[ii,jj],]-dp[dpnn$
                  neighbours[ii,1],],
94                                  dp[dpnn$neighbours[ii,kk],]-dp[dpnn$
                                      neighbours[ii,1],]))
95            ndi = ndi + 1
96           }
97         }
98        ndot[1,] = ndot[1,order(ndot[4,],decreasing = TRUE)]
99        ndot[2,] = ndot[2,order(ndot[4,],decreasing = TRUE)]
100       ndot[3,] = ndot[3,order(ndot[4,],decreasing = TRUE)]
101       ndot[4,] = ndot[4,order(ndot[4,],decreasing = TRUE)]
102
103       XYZ_new[ii,] = cbind(mean(dp[dpnn$neighbours[ii,c(1,unique(as.vector
              (ndot[c(2:3),]))[1:min(3,as.integer(NN/2))])],1]),
104                            mean(dp[dpnn$neighbours[ii,c(1,unique(as.vector
                                (ndot[c(2:3),]))[1:min(3,as.integer(NN/2))])
                                ],2]),
105                            mean(dp[dpnn$neighbours[ii,c(1,unique(as.vector
                                (ndot[c(2:3),]))[1:min(3,as.integer(NN/2))])
                                ],3]))
106     }
107   }
108 # ---------------------------------------
109   else if(method == 'Dist_to_Plane'){
110       dpn = KNNF(dp,NNN,returnall = FALSE)
111       D2P = matrix(NA,ddpts,NNN+1-3)
112       P2U = matrix(NA,ddpts,NNN+1-3)
113       for(ii in c(1:ddpts)){
114         allpts = rbind(dp[ii,],dp[dpn$neighbours[ii,],])                #
              point plus all neighbours xyz
115         allptsi = c(ii,dpn$neighbours[ii,])                             #
              point plus all neighbours indices
116         D3A = allptsi[1:3]
117         pts = NNN+1-3
118         for(jj in c(1:pts)){
119           P2U[ii,jj] = allptsi[jj+3]
120           D2P[ii,jj] = dist2plane(allpts[jj+3,], allpts[c(1:3),])  #[jj,]
121         }
122       }
123       sddist = as.matrix(apply(X = D2P, MARGIN = 1, FUN = sd, na.rm =
              TRUE))
124       meandist = as.matrix(apply(X = D2P, MARGIN = 1, FUN = mean, na.rm =
               TRUE))
125       FFP = t(apply(X = cbind(D2P,meandist,sddist), MARGIN = 1, FUN =
              FarFromPlane, PsdTimes = 1.5, NN = NN-2))
```

```
126         fardisti = as.matrix(apply(X = FFP, MARGIN = 1, FUN = match, x =
                TRUE))
127         FarPts = NA
128         ll = 1
129         for(kk in c(1:ddpts)){
130           if(!is.na(fardisti[kk,1])){
131             FarPts[ll] = P2U[kk,fardisti[kk,1]]
132             ll = ll + 1
133           }
134         }
135         # FarPts = 100
136         if(!is.na(FarPts)){
137           FarPtsU = unique(FarPts)
138           for(ii in c(1:length(FarPtsU))){
139           XYZ_new[FarPtsU[ii],1] = dp[dpn$neighbours[FarPtsU[ii],1],1]*
                  runif(1,0.99,1.01)
140           XYZ_new[FarPtsU[ii],2] = dp[dpn$neighbours[FarPtsU[ii],1],2]*
                  runif(1,0.99,1.01)
141           XYZ_new[FarPtsU[ii],3] = dp[dpn$neighbours[FarPtsU[ii],1],3]*
                  runif(1,0.99,1.01)
142           }
143           print(paste0('Nr of points changed : ',length(FarPtsU)))
144         } else {
145            print('Nr of points changed : 0')
146         }
147       }
148 # ----------------------------------------
149     else if(method == 'local_plane_angle'){
150         dpn = KNNF(dp,NNN,returnall = FALSE)
151         angles_x = matrix(NA,ddpts,NNN+2)
152         angles_x_change = matrix(NA,ddpts,NNN+1)
153         angles_y = matrix(NA,ddpts,NNN+2)
154         angles_y_change = matrix(NA,ddpts,NNN+1)
155         angles_z = matrix(NA,ddpts,NNN+2)
156         angles_z_change = matrix(NA,ddpts,NNN+1)
157         angle_change = matrix(NA,ddpts,NNN+1)
158         allptsi = matrix(NA,ddpts,NNN+1)
159         allptsonp = matrix(NA,ddpts,3)
160         for(ii in c(1:ddpts)){
161           allpts = rbind(dp[ii,],dp[dpn$neighbours[ii,],])          # point
                plus all neighbours xyz
162           allptsi[ii,] = c(ii,dpn$neighbours[ii,])                  # point
                plus all neighbours indices
163           pts = NNN+1
164           appca = prcomp(allpts)
165           p1 = apply(allpts,2,mean)
166           p2 = p1 + t(appca$rotation[,1])
167           p3 = p1 + t(appca$rotation[,2])
168           appln = getPlane(rbind(p1,p2,p3))
169           pp2 = c(appln$a,appln$b,appln$c,appln$d)
170           ppx =  c(1,0,0,0)
171           ppy =  c(0,1,0,0)
```

```
172           ppz =  c(0,0,1,0)
173           angles_x[ii,NNN+2] = get_planes_angle(ppx,pp2)
174           angles_y[ii,NNN+2] = get_planes_angle(ppy,pp2)
175           angles_z[ii,NNN+2] = get_planes_angle(ppz,pp2)
176           for(jj in c(1:pts)){
177              appca = prcomp(allpts[-jj,])
178              p1 = apply(allpts[-jj,],2,mean)
179              p2 = p1 + t(appca$rotation[,1])
180              p3 = p1 + t(appca$rotation[,2])
181              appln = getPlane(rbind(p1,p2,p3))
182              pp2 = c(appln$a,appln$b,appln$c,appln$d)
183              angles_x[ii,jj] = get_planes_angle(ppx,pp2)
184              angles_y[ii,jj] = get_planes_angle(ppy,pp2)
185              angles_z[ii,jj] = get_planes_angle(ppz,pp2)
186              angles_x_change[ii,jj] = abs(angles_x[ii,jj]-angles_x[ii,NNN
                     +2])*(180/pi)
187              angles_y_change[ii,jj] = abs(angles_y[ii,jj]-angles_y[ii,NNN
                     +2])*(180/pi)
188              angles_z_change[ii,jj] = abs(angles_z[ii,jj]-angles_z[ii,NNN
                     +2])*(180/pi)
189              angle_change[ii,jj] = angles_x_change[ii,jj] + angles_y_change
                     [ii,jj] + angles_z_change[ii,jj]
190           }
191        }
192        pts2rem = min(round(NNN/2,0),NNN-3)
193        movept = matrix(NA,ddpts,pts2rem)
194        for(ii in (1:ddpts)){
195          movept[ii,] = order(angle_change[ii,], decreasing = TRUE)[1:
                 pts2rem]
196        }
197        mpt = 0
198        pp2 = matrix(NA,ddpts,4)
199        minmaxpts = matrix(NA,ddpts,3*3)
200        for(ii in (1:ddpts)){
201          allpts = rbind(dp[ii,],dp[dpn$neighbours[ii,],])          # point
                 plus all neighbours xyz
202          allptsi[ii,] = c(ii,dpn$neighbours[ii,])                  # point
                 plus all neighbours indices
203          mps = as.vector(movept[ii,!is.na(movept[ii,])])
204            appca = prcomp(allpts[-mps,])
205            p1 = apply(allpts[-mps,],2,mean)
206            p2 = p1 + t(appca$rotation[,1])
207            p3 = p1 + t(appca$rotation[,2])
208            appln = getPlane(rbind(p1,p2,p3))
209            pp2[ii,] = c(appln$a,appln$b,appln$c,appln$d)
210            allptsonp = matrix(NA,3,3)
211            for(jj in c(1:3)){
212              allptsonp[jj,] = point_on_plane(allpts[jj,],pp2[ii,])
213            }
214            minmaxpts[ii,] = c(allptsonp[1,],allptsonp[2,],allptsonp[3,])
215            XYZ_new[ii,] = point_on_plane(allpts[1,],pp2[ii,])
216            mpt = mpt + 1
```

```
          # }
        }
print(paste0("Number of points moved = ",mpt))
   }
# ---------------------------------------
  else if(method == 'Nbr_of_Nbrs'){
# 1 = "All points inside manifold"
# 2 = "Few points outside manifold"
# 3 = "Main point outside manifold"
       dpn = KNNF(dp,NNN,returnall = FALSE)
       val_nbrsi = matrix(NA,ddpts,NNN)
       scenarios = matrix(NA,ddpts,2)
       for(ii in c(1:ddpts)){
         for(jj in c(1:NNN)){
           val_nbrsi[ii,jj] = (ii %in% dpn$neighbours[dpn$neighbours[ii,jj
               ],])
         }
           if(sum(val_nbrsi[ii,]) < NNN/2){
             scenarios[ii,1] = 3
             scenarios[ii,2] = "Main point outside manifold"
           } else if (sum(val_nbrsi[ii,]) > (NNN*0.9)) {
             scenarios[ii,1] = 1
             scenarios[ii,2] = "All points inside manifold"
           } else {
             scenarios[ii,1] = 2
             scenarios[ii,2] = "Few points outside manifold"
           }
       }
       mp = 0
       s1 = 0
       s2 = 0
       s3 = 0
       for(ii in c(1:ddpts)){
         if(scenarios[ii,1] == 3){  #"Main point outside manifold"
           # move main point to nearest common neighbor
             XYZ_new[ii,] = dp[dpn$neighbours[ii,which(val_nbrsi[ii,])
                 [1]],]*runif(1,0.99,1.01)
             mp = mp+1
             s3 = s3+1
         } else if (scenarios[ii,1] == 1) {
           # do nothing
           s1 = s1+1
         } else {   #"Few points outside manifold"
           # move main point to plane
             goodnn = which(val_nbrsi[ii,])
             allpts = rbind(dp[ii,],dp[dpn$neighbours[ii,],])[c(1,1+goodnn
                 ),]          # point plus all neighbours xyz
             allptsi = c(ii,dpn$neighbours[ii,])[c(1,1+goodnn)]
                               # point plus all neighbours indices
             appca = prcomp(allpts)
             p1 = apply(allpts,2,mean)
             p2 = p1 + t(appca$rotation[,1])
```

```
265                 p3 = p1 + t(appca$rotation[,2])
266                 appln = getPlane(rbind(p1,p2,p3))
267                 pp2 = c(appln$a,appln$b,appln$c,appln$d)
268                 XYZ_new[ii,] = point_on_plane(allpts[1,],pp2)
269                 mp = mp+1
270                 s2 = s2+1
271           }
272         }
273         print(paste0("nr of points changed = ",mp,"(",s1,",",s2,",",s3,")")
              )
274  }
275 # ----------------------------------------
276   else if(method == 'cube_density'){
277 # 1. find sparce cubes
278 # 2. find points in sparce cubes
279 # 3. move points in sparce cubes to dense cubes
280 dpn = KNNF(dp,NNN,returnall = FALSE)
281 range_x = max(dp[,1]) - min(dp[,1])
282 range_y = max(dp[,2]) - min(dp[,2])
283 range_z = max(dp[,3]) - min(dp[,3])
284 range_sum = sum(range_x,range_y,range_z)
285 ext_c = seq(from = 0.2, to = 5, length.out = 20)[6]      #1.5
286 cubesx = (range_x/(N^(1/3)))*ext_c
287 cubesy = (range_y/(N^(1/3)))*ext_c
288 cubesz = (range_z/(N^(1/3)))*ext_c
289 # -- SET 1 ------
290 x_ival_1 = seq(from = (min(dp[,1])-cubesx), by = cubesx, length.out = (
      range_x/cubesx)+1)
291 y_ival_1 = seq(from = (min(dp[,2])-cubesy), by = cubesy, length.out = (
      range_y/cubesy)+1)
292 z_ival_1 = seq(from = (min(dp[,3])-cubesz), by = cubesz, length.out = (
      range_z/cubesz)+1)
293 nr_cubes_1 = length(x_ival_1)*length(y_ival_1)*length(z_ival_1)
294 combi_1 = gtools::permutations(length(x_ival_1),3,repeats.allowed=TRUE)
295 dim(combi_1)
296 cube_lims_1 = cbind(x_ival_1[combi_1[,1]], x_ival_1[combi_1[,1]]+cubesx,
297                     y_ival_1[combi_1[,2]], y_ival_1[combi_1[,2]]+cubesy,
298                     z_ival_1[combi_1[,3]], z_ival_1[combi_1[,3]]+cubesz)
299 cube_density_1 = cbind(combi_1,apply(cube_lims_1, 1, find_density, dta =
      dp))
300 cube_points_1 = t(apply(cube_lims_1, 1, find_points, dta = dp))
301 cube_points_1i = apply(cube_points_1, 1, which, arr.ind = TRUE)
302 point_cube_1 = matrix(NA,ddpts,3)
303 point_density_1 = matrix(NA,ddpts,1)
304 for(ii in c(1:ddpts)){
305   for(jj in c(1:dim(combi_1)[1])){
306     if(length(cube_points_1i[[jj]]) > 0){
307       if(ii %in% cube_points_1i[[jj]]){
308         point_cube_1[ii,] = combi_1[jj]
309         point_density_1[ii] = cube_density_1[jj,4]
310       }
311     }
```

```
312   }
313 }
314 # -- SET 2 ------
315 x_ival_2 = seq(from = (min(dp[,1])-cubesx), by = cubesx, length.out = (
      range_x/cubesx)+1)
316 y_ival_2 = seq(from = (min(dp[,2])-cubesy), by = cubesy, length.out = (
      range_y/cubesy)+1)
317 z_ival_2 = seq(from = (min(dp[,3])-cubesz), by = cubesz, length.out = (
      range_z/cubesz)+1)
318 nr_cubes_2 = length(x_ival_2)*length(y_ival_2)*length(z_ival_2)
319 combi_2 = gtools::permutations(length(x_ival_2),3,repeats.allowed=TRUE)
320 cube_lims_2 = cbind(x_ival_2[combi_2[,1]], x_ival_2[combi_2[,1]]+(0.999999
      *cubesx),
321                     y_ival_2[combi_2[,2]], y_ival_2[combi_2[,2]]+(0.999999
                         *cubesy),
322                     z_ival_2[combi_2[,3]], z_ival_2[combi_2[,3]]+(0.999999
                         *cubesz))
323 cube_density_2 = cbind(combi_2,apply(cube_lims_2, 1, find_density, dta =
      dp))
324 cube_points_2 = t(apply(cube_lims_2, 1, find_points, dta = dp))
325 cube_points_2i = apply(cube_points_2, 1, which, arr.ind = TRUE)
326 point_cube_2 = matrix(NA,ddpts,3)
327 point_density_2 = matrix(NA,ddpts,1)
328 for(ii in c(1:ddpts)){
329   for(jj in c(1:dim(combi_2)[1])){
330     if(length(cube_points_2i[[jj]]) > 0){
331       if(ii %in% cube_points_2i[[jj]]){
332         point_cube_2[ii,] = combi_2[jj]
333         point_density_2[ii] = cube_density_2[jj,4]
334       }
335     }
336   }
337 }
338 sparse_dim = c(1,3)
339 sparse_points_1 = which(between(point_density_1,sparse_dim[1],sparse_dim
      [2]))
340 sparse_points_2 = which(between(point_density_2,sparse_dim[1],sparse_dim
      [2]))
341 sparse_point_out_1 = intersect(sparse_points_1,col_bands_out)
342 sparse_point_out_2 = intersect(sparse_points_2,col_bands_out)
343 sparse_points = intersect(sparse_points_2,sparse_points_1)
344 sparse_points_out = intersect(sparse_points,col_bands_out)
345 print(paste0("nr of points changed = ",length(sparse_points)))
346 sparse_points_missing = setdiff(col_bands_out,sparse_points)
347 dp[sparse_points,] = dp[dpn$neighbours[sparse_points,1],]*runif
      (1,0.99,1.01)
348 XYZ_new = dp
349 }
350   return(XYZ_new)
351 }
```

C Appendix C - R Code of Supporting Functions

```
KNNF = function(datapoints, NN, returnall = FALSE){
p = datapoints
DP = dim(p)[1]
D = matrix(NA,DP,DP)
edges = D
for(ii in c(1:DP)){
  for(jj in c(1:DP)){
    D[ii,jj] = sqrt((p[jj,1]-p[ii,1])^2 + (p[jj,2]-p[ii,2])^2 + (p[jj,3]-p
        [ii,3])^2)
  }
}
DO = matrix(NA,DP,DP)
for(ii in c(1:DP)){
  DO[ii,] = order(D[ii,])
  edges[ii,] = D[ii,order(D[ii,])]
}
  if (returnall == TRUE){
    return(list(edges = edges[,1:DP], neighbours = DO[,1:DP], k = NN))
    } else {
    return(list(edges = edges[,1:NN+1], neighbours = DO[,1:NN+1], k = NN))
    }
}

# ------------------------------------------------------------------

MakeDistNextMatrix = function(datapoints, K, InfAfterK = TRUE){
testpknn = KNNF(datapoints,K,!InfAfterK)
pts = length(testpknn$edges[1,])
apts = length(testpknn$edges[,1])
dist = matrix(Inf,apts,apts)
nexts = matrix(NA,apts,apts)
for(aa in c(1:apts)){
  for(bb in c(1:pts)){
      dist[aa,testpknn$neighbours[aa,bb]] = testpknn$edges[aa,bb]
      nexts[aa,testpknn$neighbours[aa,bb]] = testpknn$neighbours[aa,bb]
    }
  }
for(aa in c(1:apts)){
      dist[aa,aa] = 0
      nexts[aa,aa] = aa
```

```
      }
return(list(dist = dist, nexts = nexts, apts = apts, pts = pts))
}

# ------------------------------------------------------------------

Floyd_Orig = function(datapoints, K){
MM = MakeDistNextMatrix(datapoints, K)
nexts_init = MM$nexts
adj = 0
for(kk in c(1:MM$apts)){
  for(ii in c(1:MM$apts)){
    for(jj in c(1:MM$apts)){
        if(MM$dist[ii,jj] > MM$dist[ii,kk] + MM$dist[kk,jj]){
          adj = adj + 1
          MM$dist[ii,jj] = MM$dist[ii,kk] + MM$dist[kk,jj]
          MM$nexts[ii,jj] = MM$nexts[ii,kk]
        }
      }
    }
  }
paths = matrix(NA,MM$apts,MM$apts)
for(xx in c(1:MM$apts)){
  for(yy in c(1:MM$apts)){
    if(is.na(MM$nexts[xx,yy])){
      paths[xx,yy] = xx
      paths[xx,yy] = paste(paths[xx,yy],yy)
    }
    else{
      paths[xx,yy] = xx
      xn = xx
        while (xn != yy){
         xn = MM$nexts[xn,yy]
         paths[xx,yy] = paste(paths[xx,yy],xn)
        }
    }
  }
}
return(list(paths = paths,dist = MM$dist))
}

# ------------------------------------------------------------------

getPlane = function(three_points){
  tp = three_points
  x1 = tp[1,1]
  y1 = tp[1,2]
  z1 = tp[1,3]
  x2 = tp[2,1]
  y2 = tp[2,2]
  z2 = tp[2,3]
  x3 = tp[3,1]
```

```
  y3 = tp[3,2]
  z3 = tp[3,3]
  a1 = x2 - x1
  b1 = y2 - y1
  c1 = z2 - z1
  a2 = x3 - x1
  b2 = y3 - y1
  c2 = z3 - z1
  a = b1 * c2 - b2 * c1
  b = a2 * c1 - a1 * c2
  c = a1 * b2 - b1 * a2
  d = (- a * x1 - b * y1 - c * z1)
  return(list(a=a, b=b, c=c, d=d))
}

# ------------------------------------------------------------------

dist2plane = function(datapoint,three_points){
  dp = datapoint
  tp = three_points
  pn = getPlane(tp)
  return((abs(pn$a*dp[1]+pn$b*dp[2]+pn$c*dp[3]+pn$d))/(sqrt(((pn$a)^2)+((
      pn$b)^2)+((pn$c)^2))))
}

# ------------------------------------------------------------------

FarFromPlane = function(PdPmPsd,PsdTimes,NN){
          return(PdPmPsd[1:NN] > PdPmPsd[NN+1]+(PdPmPsd[NN+2]*PsdTimes))
}

# ------------------------------------------------------------------

find_points = function(dta, cube_lims){
  in_pts = FALSE  #vector(FALSE,dim(dta)[1])
  for(ii in c(1:dim(dta)[1])){
    if((dta[ii,1] >= cube_lims[1]) & (dta[ii,1] < cube_lims[2]) &
       (dta[ii,2] >= cube_lims[3]) & (dta[ii,2] < cube_lims[4]) &
       (dta[ii,3] >= cube_lims[5]) & (dta[ii,3] < cube_lims[6])){
      in_pts[ii] = TRUE
    } else {
      in_pts[ii] = FALSE
    }
  }
  return(in_pts)
}

# ------------------------------------------------------------------

find_density = function(dta, cube_lims){
  # print(dim(dta))
  # print(dta)
```

```
143   cnt_pts = 0
144   for(ii in c(1:dim(dta)[1])){
145     if((dta[ii,1] >= cube_lims[1]) & (dta[ii,1] < cube_lims[2]) &
146        (dta[ii,2] >= cube_lims[3]) & (dta[ii,2] < cube_lims[4]) &
147        (dta[ii,3] >= cube_lims[5]) & (dta[ii,3] < cube_lims[6])){
148       cnt_pts = cnt_pts +1
149     }
150   }
151   return(cnt_pts)
152 }
153
154 # ------------------------------------------------------------------
155
156 range_val = function(vect){return(max(vect) - min(vect))}
157 range_out = function(vect){
158   return(max(abs(vect[1] - max(vect[-1])),abs(vect[1] - min(vect[-1]))))
159 }
160
161 # ------------------------------------------------------------------
162
163 IQR_vals = function(vect, out_factor){
164   IntQrtlRange = quantile(vect, c(.75)) - quantile(vect, c(.25))
165   top_val = quantile(vect, c(.75)) + out_factor*IntQrtlRange
166   bot_val = quantile(vect, c(.25)) - out_factor*IntQrtlRange
167   return(list(vect = vect, out_factor = out_factor, IntQrtlRange =
        IntQrtlRange, top_val = top_val, bot_val = bot_val))
168 }
```

D Appendix D - R Code of Stochastic Any-Paths Algorithm

```
paths_ap = function(res, minPass = 1, maxPass = 10)
 {
   N   = dim(res$edges)[1]
   set   = 1:N
   setI  = c()
   setJ  = c()
   Dg = matrix(Inf,N,N)
   for(i in 1:N)
   {
       Dg[i,res$neighbours[i,]] = res$edges[i,]
       setI = c(setI,rep(i,N)[-res$neighbours[i,]])
       setJ = c(setJ,set[-res$neighbours[i,]])
   }
   diag(Dg) = 0
   setR = 1:length(setI)
   setR = setR[-which(abs(setI - setJ)==0)]
   asp = allShortestPaths(Dg)
   kk   = 0
   M    = matrix(rep(1:N,N),N,N,byrow = F)
   Path = matrix(paste0(M,',',t(M)),N,N,byrow = F)
   Path[Dg==Inf] = ''
   check = any(is.infinite(Dg))
   while((check|(kk<minPass))&kk<maxPass)
   {
     Dtemp = Dg
     Pathtemp = Path
     rmr = c()
     for(r in setR)
     {
         i = setI[r]
         j = setJ[r]
         prbs = 1/(Dg[i,]+Dg[,j])     #ifelse((Dg[i,]+Dg[,j])==0,1,(Dg[i,]+
             Dg[,j]))
         prbs[is.na(prbs)] = 0
         prbs[i] = 0
         prbs[j] = 0
         if(max(prbs,na.rm = TRUE) != 0)
         {
           l = sample(1:N, 1, prob = prbs != 0)
           if(Dg[i,l]+Dg[l,j] < Dg[i,j])
```

```
            {
              Dtemp[i,j] = Dg[i,l]+Dg[l,j]
              Pathtemp[i,j]  = paste0(Path[i,l],'_',Path[l,j])
            }
        }
      }
      kk     = kk + 1
      print(paste0("pass = ",kk))
      Dg     = Dtemp
      check = any(is.infinite(Dg))
      Path    = Pathtemp
    }
    return(list(Dg = Dg, asp = asp, Path = Path, passes = kk))
  }
```

E Appendix E - Example of Modeling on Realworld Dataset

```
# Data ------
set.seed(444)
rm(list = setdiff(ls(envir = .GlobalEnv), lsf.str(envir = .GlobalEnv)),
    envir = .GlobalEnv)
churn_data_raw = read.csv("telecom_churn_data.csv")
dim(churn_data_raw)  #[1] 99999   226
churn_data_raw[is.na(churn_data_raw)] = 0
Not_used = c('mobile_number','last_date_of_month_6','last_date_of_month_7'
    ,'last_date_of_month_8','last_date_of_month_9',
          'date_of_last_rech_6','date_of_last_rech_7','date_of_last_rech_8
              ','date_of_last_rech_9',
          'date_of_last_rech_data_6','date_of_last_rech_data_7','date_of_
              last_rech_data_8','date_of_last_rech_data_9')
churn_data_used = churn_data_raw[,!(names(churn_data_raw) %in% Not_used)]
cvar = apply(X = churn_data_used, MARGIN = 2, FUN = var, na.rm = TRUE)
no_var = which(cvar==0)
churn_data_var = churn_data_used[,-no_var]
dim(churn_data_var)  #[1] 99999   209
churn = c('total_ic_mou_9','total_og_mou_9','vol_2g_mb_9','vol_3g_mb_9')
churn_data_Y = matrix(ifelse(apply(churn_data_var[,churn],1,sum)==0,"YES",
    "NO"),ncol = 1)       #churn = 1
colnames(churn_data_Y) = 'churn_flag'
churn_data_X = churn_data_var[,!(names(churn_data_var) %in% churn)]
dim(churn_data_X)  #[1] 99999   197
churn_sdata_X = churn_data_X/apply(churn_data_X, 2, max)
datai = seq.int(from = 1, to = dim(churn_data_X)[1])
train_size = 1000
test_size  = 1000
traini = sample.int(dim(churn_data_X)[1],size = train_size)
testi = sample(datai[-traini],size = test_size)
# ---split  ----------------------------
tnt = cbind(churn_data_Y,churn_sdata_X)
tntx = tnt[,-1]
tnty = matrix(tnt[,1],ncol = 1)
train = tnt[traini,]
trainx = train[,-1]
trainy = matrix(train[,1],ncol = 1)
test = tnt[testi,]
testx = test[,-1]
testy = matrix(test[,1],ncol = 1)
```

```
# ------------------------------------------------------------------

# Models ------
# ------------------------------------------------------------------
# ----- dataset as is ---------
f_train <- data.frame(churn_flag = factor(trainy), trainx)
f_test <- data.frame(churn_flag = factor(testy), testx)
grid_rbf = expand.grid(weight = c(0.0001, 0.001, 0.01),
                       cost     = c(0.0001, 0.001, 0.01))
fitControl <- trainControl(method = "cv", number = 5)
fit.rbf2 <- caret::train(churn_flag ~ ., data = f_train,  metric = "Accuracy", method = "svmLinearWeights",
                         trControl = fitControl, tuneGrid = grid_rbf,
                             verbose = FALSE, preProcess = c("center", "scale") )
fit.rbf2$bestTune$weight
fit.rbf2$bestTune$cost
tr_cv_rbf = predict(fit.rbf2, type = 'raw')
AccChrnLn1_train_AsIs = sum(tr_cv_rbf == f_train$churn_flag) / nrow(f_train)
eval_cv_rbf <- predict(fit.rbf2, newdata = f_test, type = 'raw')
AccChrnLn1_test_AsIs = sum(eval_cv_rbf == f_test$churn_flag) / nrow(f_test)
AccChrnLn1_train_AsIs
AccChrnLn1_test_AsIs

# ============================================================
# ---- using PCA on original data ----------------------------
# dddd = prcomp(trainx)
# plot(cumsum(dddd$sdev^2)/sum(dddd$sdev^2), type = "b")
# aaaa = cumsum(dddd$sdev^2)/sum(dddd$sdev^2)
# aaaa[50]
PCs = 50
pcatrainx = prcomp(trainx)$x[,c(1:PCs)]
pcatrain = data.frame(churn_flag = factor(trainy),pcatrainx)
pcatestx = scale(testx, prcomp(trainx)$center, prcomp(trainx)$scale) %*% prcomp(trainx)$rotation
pcatestx = pcatestx[,c(1:PCs)]
pcatest = data.frame(churn_flag = factor(testy),pcatestx)
grid_rbf = expand.grid(weight = c(0.0001, 0.001, 0.01),
                       cost     = c(0.0001, 0.001, 0.01))
fitControl <- trainControl(method = "cv", number = 5)
fit.rbf2 <- caret::train(churn_flag ~ ., data = pcatrain,  metric = "Accuracy", method = "svmLinearWeights",
                         trControl = fitControl, tuneGrid = grid_rbf,
                             verbose = FALSE, preProcess = c("center", "scale") )
fit.rbf2$bestTune$weight
fit.rbf2$bestTune$cost
tr_cv_rbf = predict(fit.rbf2, type = 'raw')
AccChrnLn2_train_PCA = sum(tr_cv_rbf == pcatrain$churn_flag) / nrow(pcatrain)
```

```
eval_cv_rbf <- predict(fit.rbf2, newdata = pcatest, type = 'raw')
AccChrnLn2_test_PCA = sum(eval_cv_rbf == pcatest$churn_flag) / nrow(
    pcatest)
AccChrnLn2_train_PCA
AccChrnLn2_test_PCA

# ============================================================
# ---- using PCA on SP -----------------------------
K = 50
excl=c(8,20,32,35,45,104,123,133,193,197,202,215,264,300,349,381,386,
  391,409,424,428,452,480,481,491,496,511,529,530,543,544,561,563,571,
  632,635,647,704,717,753,778,813,833,879,885,886,940,951,994)
mtrainx = as.matrix(trainx)[-excl,]
res = k_nn(mtrainx,mtrainx,K)
asps = paths_sp(res)
trainxasp = asps$Dg
print(paste0("Nr unreachable pts : ",
             length(unique(c(which(apply(X = trainxasp, MARGIN = 1, FUN =
                 function(x) sum(is.na(x)))>100),
                             which(apply(X = trainxasp, MARGIN = 1, FUN =
                                 function(x) sum(is.infinite(x)))>100),
                             which(apply(X = trainxasp, MARGIN = 2, FUN =
                                 function(x) sum(is.na(x)))>100),
                             which(apply(X = trainxasp, MARGIN = 2, FUN =
                                 function(x) sum(is.infinite(x)))>100))))))
which(apply(X = trainxasp, MARGIN = 1, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = trainxasp, MARGIN = 1, FUN = function(x) sum(is.infinite(x
    )))>0)
which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.infinite(x
    )))>0)
length(which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.na(
    x)))>0))
pctx = prcomp(trainxasp, center = TRUE, scale. = TRUE)
pcatrainx = pctx$x[,c(1:PCs)]
pcatrain = data.frame(churn_flag = factor(trainy[-excl,]),pcatrainx)
# -----------------
K = 25
excl=c(39,55,94,200,203,226,232,237,243,249,320,325,331,382,391,405,
  443,467,489,497,508,523,529,531,582,591,603,645,646,654,677,678,702,
  708,744,756,763,772,786,790,793,829,853,854,864,886,912,928,929,942)
mtestx = as.matrix(testx)[-excl,]
res = k_nn(mtestx,mtestx,K)
asps = paths_sp(res)
testxasp = asps$Dg
which(apply(X = testxasp, MARGIN = 1, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = testxasp, MARGIN = 1, FUN = function(x) sum(is.infinite(x)
    ))>0)
which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.infinite(x)
    ))>1)
```

```
length(which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.na(x
    )))>0))
pcatestx = scale(testxasp, pctx$center[1:dim(testxasp)[2]], pctx$scale[1:
    dim(testxasp)[2]]) %*% pctx$rotation[1:dim(testxasp)[2],]
pcatestx = pcatestx[,c(1:PCs)]
pcatest = data.frame(churn_flag = factor(testy[-excl,]),pcatestx)
grid_rbf = expand.grid(weight = c(0.0001, 0.001, 0.01),
                       cost     = c(0.0001, 0.001, 0.01))
fitControl <- trainControl(method = "cv", number = 5)
fit.rbf2 <- caret::train(churn_flag ~ ., data = pcatrain,  metric = "
    Accuracy", method = "svmLinearWeights",
                         trControl = fitControl, tuneGrid = grid_rbf,
                             verbose = FALSE, preProcess = c("center", "
                             scale") )
fit.rbf2$bestTune$weight
fit.rbf2$bestTune$cost
tr_cv_rbf = predict(fit.rbf2, type = 'raw')
AccChrnLn3_train_SP = sum(tr_cv_rbf == pcatrain$churn_flag) / nrow(
    pcatrain)
eval_cv_rbf <- predict(fit.rbf2, newdata = pcatest, type = 'raw')
AccChrnLn3_test_SP = sum(eval_cv_rbf == pcatest$churn_flag) / nrow(pcatest
    )
AccChrnLn3_train_SP
AccChrnLn3_test_SP

# ========================================================================
# ---- using PCA on SP with "knn lim"  -----------------------------
K = 50
dp = as.matrix(trainx)
ddpts = dim(trainx)[1]
NNN = K
XYZ_new = dp
alfa = 1.3
    dpn = k_nn(dp,dp,NNN)
    ptch = 0
    for(ii in c(1:ddpts)){
      mn_edge = mean(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      sd_edge = sd(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      for(jj in c(1:NNN)){
        nnch = 0
          if(dpn$edges[ii,jj] > alfa * mn_edge){
            dpn$neighbours[ii,jj] = NA
            nnch = nnch + 1
          }
      }
        XYZ_new[ii,] = apply(dp[dpn$neighbours[ii,],],2,mean,na.rm = TRUE)
        if(nnch > 0){
          ptch = ptch + 1
        }
    }
    print(paste0("nr of points changed = ",ptch))
Kp = 40
```

```
res = k_nn(XYZ_new,XYZ_new,Kp)
asps = paths_sp(res)
trainxasp = asps$Dg
which(apply(X = trainxasp, MARGIN = 1, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = trainxasp, MARGIN = 1, FUN = function(x) sum(is.infinite(x
    )))>0)
which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.infinite(x
    )))>0)
length(which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.na(
    x)))>0))
pctx = prcomp(trainxasp, center = TRUE, scale. = TRUE)
pcatrainx = pctx$x[,c(1:PCs)]
pcatrain = data.frame(churn_flag = factor(trainy),pcatrainx)  #[-excl,]
# --------------------------
K = 50
dp = as.matrix(testx)
ddpts = dim(testx)[1]
NNN = K
XYZ_new = dp
alfa = 1.3
    dpn = k_nn(dp,dp,NNN)
    ptch = 0
    for(ii in c(1:ddpts)){
      mn_edge = mean(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      sd_edge = sd(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      for(jj in c(1:NNN)){
        nnch = 0
          if(dpn$edges[ii,jj] > alfa * mn_edge){
            dpn$neighbours[ii,jj] = NA
            nnch = nnch + 1
          }
      }
        XYZ_new[ii,] = apply(dp[dpn$neighbours[ii,],],2,mean,na.rm = TRUE)
        if(nnch > 0){
          ptch = ptch + 1
        }
    }
    print(paste0("nr of points changed = ",ptch))
Kp = 40
res = k_nn(XYZ_new,XYZ_new,Kp)
asps = paths_sp(res)
testxasp = asps$Dg
which(apply(X = testxasp, MARGIN = 1, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = testxasp, MARGIN = 1, FUN = function(x) sum(is.infinite(x)
    ))>0)
which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.infinite(x)
    ))>0)
length(which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.na(x
    )))>0))
pcatestx = scale(testxasp, pctx$center, pctx$scale) %*% pctx$rotation
```

```
pcatestx = pcatestx[,c(1:PCs)]
pcatest = data.frame(churn_flag = factor(testy),pcatestx)  #[-excl,]
grid_rbf = expand.grid(weight = c(0.0001, 0.001, 0.01),
                       cost     = c(0.0001, 0.001, 0.01))
fitControl <- trainControl(method = "cv", number = 5)
fit.rbf2 <- caret::train(churn_flag ~ ., data = pcatrain,  metric = "Accuracy", method = "svmLinearWeights",
                         trControl = fitControl, tuneGrid = grid_rbf,
                             verbose = FALSE, preProcess = c("center", "scale") )
fit.rbf2$bestTune$weight
fit.rbf2$bestTune$cost
tr_cv_rbf = predict(fit.rbf2, type = 'raw')
AccChrnLn4_train_SPNN = sum(tr_cv_rbf == pcatrain$churn_flag) / nrow(pcatrain)
eval_cv_rbf <- predict(fit.rbf2, newdata = pcatest, type = 'raw')
AccChrnLn4_test_SPNN = sum(eval_cv_rbf == pcatest$churn_flag) / nrow(pcatest)
AccChrnLn4_test_SPNN
AccChrnLn4_test_SPNN

# ================================================================
# ---- using PCA on AP with "knn lim"  -------------------------
K = 50
dp = as.matrix(trainx)
ddpts = dim(trainx)[1]
NNN = K
XYZ_new = dp
alfa = 1.3
    dpn = k_nn(dp,dp,NNN)
    ptch = 0
    for(ii in c(1:ddpts)){
      mn_edge = mean(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      sd_edge = sd(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      for(jj in c(1:NNN)){
        nnch = 0
          if(dpn$edges[ii,jj] > alfa * mn_edge){
            dpn$neighbours[ii,jj] = NA
            nnch = nnch + 1
          }
      }
        XYZ_new[ii,] = apply(dp[dpn$neighbours[ii,],],2,mean,na.rm = TRUE)
        if(nnch > 0){
          ptch = ptch + 1
        }
    }
    print(paste0("nr of points changed = ",ptch))
Kp = 40
res = k_nn(XYZ_new,XYZ_new,Kp)
asps = paths_ap(res)
trainxasp = asps$Dg
which(apply(X = trainxasp, MARGIN = 1, FUN = function(x) sum(is.na(x)))>0)
```

```
which(apply(X = trainxasp, MARGIN = 1, FUN = function(x) sum(is.infinite(x
    )))>0)
which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.infinite(x
    )))>0)
length(which(apply(X = trainxasp, MARGIN = 2, FUN = function(x) sum(is.na(
    x)))>0))
pctx = prcomp(trainxasp, center = TRUE, scale. = TRUE)
pcatrainx = pctx$x[,c(1:PCs)]
pcatrain = data.frame(churn_flag = factor(trainy),pcatrainx)  #[-excl,]
# --------------------------
K = 50
dp = as.matrix(testx)
ddpts = dim(testx)[1]
NNN = K
XYZ_new = dp
alfa = 1.3
    dpn = k_nn(dp,dp,NNN)
    ptch = 0
    for(ii in c(1:ddpts)){
      mn_edge = mean(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      sd_edge = sd(as.vector(dpn$edges[ii,]),na.rm = TRUE)
      for(jj in c(1:NNN)){
        nnch = 0
          if(dpn$edges[ii,jj] > alfa * mn_edge){
            dpn$neighbours[ii,jj] = NA
            nnch = nnch + 1
          }
      }
        XYZ_new[ii,] = apply(dp[dpn$neighbours[ii,],],2,mean,na.rm = TRUE)
        if(nnch > 0){
          ptch = ptch + 1
        }
    }
    print(paste0("nr of points changed = ",ptch))
Kp = 40
res = k_nn(XYZ_new,XYZ_new,Kp)
asps = paths_ap(res)
testxasp = asps$Dg
which(apply(X = testxasp, MARGIN = 1, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = testxasp, MARGIN = 1, FUN = function(x) sum(is.infinite(x)
    ))>0)
which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.na(x)))>0)
which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.infinite(x)
    ))>0)
length(which(apply(X = testxasp, MARGIN = 2, FUN = function(x) sum(is.na(x
    )))>0))
pcatestx = scale(testxasp, pctx$center, pctx$scale) %*% pctx$rotation
pcatestx = pcatestx[,c(1:PCs)]
pcatest = data.frame(churn_flag = factor(testy),pcatestx)  #[-excl,]
grid_rbf = expand.grid(weight = c(0.0001, 0.001, 0.01),
                       cost   = c(0.0001, 0.001, 0.01))
```

```
fitControl <- trainControl(method = "cv", number = 5)
fit.rbf2 <- caret::train(churn_flag ~ ., data = pcatrain,  metric = "Accuracy", method = "svmLinearWeights",
                         trControl = fitControl, tuneGrid = grid_rbf,
                             verbose = FALSE, preProcess = c("center", "scale") )
fit.rbf2$bestTune$weight
fit.rbf2$bestTune$cost
tr_cv_rbf = predict(fit.rbf2, type = 'raw')
AccChrnLn5_train_APNN = sum(tr_cv_rbf == pcatrain$churn_flag) / nrow(pcatrain)
eval_cv_rbf <- predict(fit.rbf2, newdata = pcatest, type = 'raw')
AccChrnLn5_test_APNN = sum(eval_cv_rbf == pcatest$churn_flag) / nrow(pcatest)
AccChrnLn5_train_APNN
AccChrnLn5_test_APNN
# ---------------------------------
ChurnLin_tab = rbind(c('AsIs',round(AccChrnLn1_train_AsIs*100,0),round(AccChrnLn1_test_AsIs*100,0),0,0),
            c('PCA',round(AccChrnLn2_train_PCA*100,0),round(AccChrnLn2_test_PCA*100,0),0,0),
            c('SP',round(AccChrnLn3_train_SP*100,0),round(AccChrnLn3_test_SP*100,0),
                 round(100*(AccChrnLn3_train_SP-AccChrnLn2_train_PCA)/AccChrnLn2_train_PCA,1),
                 round(100*(AccChrnLn3_test_SP-AccChrnLn2_test_PCA)/AccChrnLn2_test_PCA,1)),
            c('SPNN',round(AccChrnLn4_train_SPNN*100,0),round(AccChrnLn4_test_SPNN*100,0),
                 round(100*(AccChrnLn4_train_SPNN-AccChrnLn3_train_SP)/AccChrnLn3_train_SP,1),
                 round(100*(AccChrnLn4_test_SPNN-AccChrnLn3_test_SP)/AccChrnLn3_test_SP,1)),
            c('APNN',round(AccChrnLn5_train_APNN*100,0),round(AccChrnLn5_test_APNN*100,0),
                 round(100*(AccChrnLn5_train_APNN-AccChrnLn3_train_SP)/AccChrnLn3_train_SP,1),
                 round(100*(AccChrnLn5_test_APNN-AccChrnLn3_test_SP)/AccChrnLn3_test_SP,1)))
```

F Appendix F - Summary of Terminology Used in the Dissertation

PCA **P**rincipal **C**omponent **A**nalysis
PCA is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components by defining a new orthogonal coordinate system that optimally describes variance in a single dataset.

NMF **N**on-negative **M**atrix **F**actorization
NMF is an algorithm which factorizes a matrix into two matrices with no negative elements, where the one matrix provides a representation of the basis elements (building blocks), and the other the coordinates of how to reconstruct the approximation of the original matrix.

LDA **L**inear **D**iscriminant **A**nalysis
LDA projects data from a D dimensional feature space down to a D' (D>D') dimensional space in a way to **maximize the variability between the classes** and reducing the variability within the classes.

GDA **G**eneralised **D**iscriminant **A**nalysis
GDA, similar to LDA, uses the kernel function operator to map the input vectors into high-dimensional feature space. The objective of GDA is to project the features into a lower-dimensional space in order to find a feature space in which variables are **non-linearly** related to the input space.

t-SNE **t**-distributed **S**tochastic **N**eighbour **E**mbedding
t-SNE is similar to PCA, but has the distinction that it tries to preserve only **small pairwise distances** or local similarities, whereas PCA is concerned with preserving large pairwise distances to maximize variance.

MDS **M**ulti **D**imensional **S**caling

MDS is a family of statistical methods that focus on creating mappings of items based on distance or dissimilarity. It includes Metric Multidimensional Scaling (or PCA), Nonmetric Multidimensional Scaling, Individual Differences Scaling and Multidimensional Analysis of Preference.

Zero-variance

If a given set of data values has zero variance, then it means that the data values are constant. The data values consist of the same number repeated a certain number of times.

Bibliography

Mukund Balasubramanian and Eric L. Schwartz. The isomap algorithm and topological stability. *Science*, 295(7a), 2002. doi: DOI:10.1126/science.295.5552.7a.

David Demers and Garrison Cottrell Y. Non-linear dimensionality reduction. In *Advances in Neural Information Processing Systems 5*, pages 580–587. Morgan Kaufmann, 1993.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959. doi: 10.1007/bf01386390.

Francois Evert. MDS_TH1_CODE. URL https://github.com/francois-evert/MDS_TH1_CODE. Accessed 2022-04-12.

Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *National Science Review*, 1(2):293–314, 02 2014. ISSN 2095-5138. doi: 10.1093/nsr/nwt032. URL https://doi.org/10.1093/nsr/nwt032.

Robert Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345, 06 1962.

Anthony Goldbloom. Find Open Datasets and Machine Learning Projects on Kaggle. URL https://www.kaggle.com/datasets. Accessed 2022-04-12.

Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing, Second Edition*. Addison Wesley, 2003. ISBN 0-201-64865-2.

Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/tssc.1968.300136. URL https://doi.org/10.1109/tssc.1968.300136.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370.

Anna A Kostina, Plamen M Tzvetkov, and Andrey N Serov. Investigation of the method of rms measurement based on moving averaging. In *2020 55th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pages 235–238. IEEE, 2020.

Guido Kraemer, Markus Reichstein, and Miguel D. Mahecha. dimRed and coRanking—Unifying Dimensionality Reduction in R. *The R Journal*, 10(1):342–358, 2018. URL https://journal.r-project.org/archive/2018/RJ-2018-039/index.html.

John Aldo Lee and Michel Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29–53, 2005. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2004.11.042. URL https://www.sciencedirect.com/science/article/pii/S0925231205001104. Geometrical Methods in Neural Networks and Learning.

Li Jing Mahwish Yousaf, Tanzeel U. Rehman. An extended isomap approach for nonlinear dimension reduction. *SN Computer Science*, 1(160), 05 2020. ISSN 2661-8907. doi: 10.1007/s42979-020-00179-y. URL https://doi.org/10.1007/s42979-020-00179-y.

David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2020. URL https://CRAN.R-project.org/package=e1071. R package version 1.7-4.

Jari Oksanen, F. Guillaume Blanchet, Michael Friendly, Roeland Kindt, Pierre Legendre, Dan McGlinn, Peter R. Minchin, R. B. O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens, Eduard Szoecs, and Helene Wagner. *vegan: Community Ecology Package*, 2020. URL https://CRAN.R-project.org/package=vegan. R package version 2.5-7.

C. Orsenigo and C. Vercellis. Linear versus nonlinear dimensionality reduction for banks' credit rating prediction. *Knowledge-Based Systems*, 47:14–22, 2013. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2013.03.001. URL https://www.sciencedirect.com/science/article/pii/S0950705113000816.

Akanksha Singh and Pramod Kumar Mishra. Performance analysis of floyd warshall algorithm vs rectangular algorithm. *International Journal of Computer Applications*, 107(16), 2014.

Sotiris Tasoulis, Nicos G. Pavlidis, and Teemu Roos. Nonlinear dimensionality reduction for clustering. *Pattern Recognition*, 107:107508, 2020. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2020.107508. URL https://www.sciencedirect.com/science/article/pii/S0031320320303113.

Joshua Tenenbaum, Vin de Silva, and John Langford. A global geometric framework for nonlinear dimensionality reduction. *SCIENCE*, 290(1):2319, 2000.

Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative review. *J Mach Learn Res*, 10:66–71, 2009.

Jiaoyun Yang, Haipeng Wang, Huitong Ding, Ning An, and Gil Alterovitz. Nonlinear dimensionality reduction methods for synthetic biology biobricks' visualization. *BMC Bioinformatics*, 18(1), 2017. doi: 10.1186/s12859-017-1484-4.