Visual Representation of Spatial Relationships in Higher Dimensions

James Weichert

June, 2018

Higher Level Mathematics - Mrs. Jayson & Mr. Snyder

Contents

Part I
Introduction

Our world is one of ever-evolving information. In this age of big data, where everything can be distilled down to numerical values that fall neatly into labels or variables, it has become of increasing interest to find connections and relationships among that enormous dataset. Tools and processes such as mathematical computer models (e.g. logistic or linear regression), statistical analysis, and machine learning driven by neural networks have enabled the calculation of such relationships at speeds exponentially faster than human calculation could ever yield.

Despite this wealth of data, there remains a major barrier to understanding data with numerous variables (that is to say, with numerous moving parts). Namely, as the number of variables in a dataset extends beyond two or three, our ability to visualize that data in an intuitive manner pushes up against the boundaries of human understanding in our three-dimensional world. Because everything in the universe exists strictly in three-dimensions, it is impossible to simplify data in higher-dimensions (i.e. beyond three variables) down to a three- or two-dimensional graphical representation that maintains all of the properties and spatial relationships that data had in its original dimensionality. Instead, datasets in higher dimensions must first be transformed and manipulated before being represented visually. This often comes at the cost of a loss in the integrity of the relationships between data points, whereby the reduction in dimensionality eliminates connections among data points. Therefore, alternative visualization techniques are needed in order to intuitively visualize data in higher dimensions while maintaining the integrity of the spatial relationships between points in the dataset.

The purpose of this paper is to explore such methods of visual representation to arrive at a visualization method which provides for an understandable view of higher-dimensional data all the while allowing for the interpretation of that data by showing the relative relationships among data points.

As someone who has always been fascinated by how data can aid in our understanding of the physical world, I've always held a particular interest for how graphs and mathematical visualizations can be beneficial for interpreting data. Being a very visual person, I often find myself needing to understand a mathematical concept visually before I can fully understand it theoretically. For example, to this day I need to draw out the unit circle in order to make sure that my trigonometry is correct and always need to think of the derivative as the slope of a function and the integral as the area under a curve.

Lately, I've been working a lot with artificial intelligence and machine learning in other academic projects, which often involves this notion of working with 'big data'. The central idea behind big data is that the number of variables available to measure for a particular object or data point is so immense, that it is extremely difficult to wrap your head around just how complex the data is. This presents numerous problems for me as someone interested in data science, the greatest of which is the question of how I can understand what I'm doing intuitively if I don't have a visual way to help me learn about the mathematical properties of the data I'm looking at. Furthermore, explaining my work to other people who don't

understand machine learning becomes exponentially harder when I can't show them what it is I'm doing in a visual manner.

As such, I am not only academically interested in understanding how to visualize higher-dimensional relationships, but am also personally invested in this topic, as it can only aid me in better understanding other mathematical and computational work in the fields of data science and computer science. Additionally, being able to show people a tangible visual model of my work is a far better alternative for allowing lay persons to understand and appreciate such work than the phrase *I work with big data*.

Part II
Conventional Visualizations

1 Visualization in Two and Three Dimensions

Before I discuss just how higher-dimensional data can be projected onto a two- or three-dimensional graphical representation, I think it is necessary to highlight the conventions and properties of such lower-dimensional representations, especially with regard to why they are so powerful. Namely, we must answer the question *why do we want to represent data on a graph?* before we can answer the question *how do we represent data on a graph?*

1.1 Rectangular Coordinate System

Unless otherwise stated, a "two- or three-dimensional graphical visualization," is a graph of a rectangular (Cartesian) coordinate system with each data point plotted as a point on that graph. This is conventionally achieved by mapping each variable of the data to an axis on the graph. In two dimensions, the first variable is mapped to the x-axis while the second variable is mapped to the y-axis. In three dimensions, the first two variables are mapped to the x- and y-axes, respectively, while the third variable is mapped to the z-axis.

This lower-dimensional visualization is so powerful[1] in part because the visualization allows us to rationalize relationships as the 'the effect that one variable has on another'. I find that this notion of relationships connects neatly to how we learn about functions in secondary school, where we're taught that a function transforms one number into another number using a certain formula. In this way, it is quite easy to understand that the result of moving to the right on a two-dimensional graph (increasing the value of x) is the change in the height (y-value) of the function at that point. Thus, the process of analyzing graphical relationships boils down to comparing the relative positions of two points on the graph. This reasoning proves especially useful for projections of higher-dimensional data onto a lower-dimensional graph, because the data is rarely positioned relative to the values of its variables, but rather relative to the relationships among the data points that existed in the higher dimension.

[1] In terms of the analytical insight it provides

1.2 Common Visualizations

Some common visualization techniques, which aim to represent the entirety of the data in all of its dimensions, include: *Radar Charts*, *Scatter Plot Matrices*, and *Parallell Coordinate Systems*[2] (Grinstein et al.).

Part III
Representating Spatial Relationships

2 Linear Dimensionality Reduction

The conventional visualization techniques mentioned above had the goal of simply mapping the higher-dimensional data set onto a two-dimensional graph space without eliminating any of the variables (dimensions) in the dataset. They achieve this by defining certain areas of that two-dimensional space as representing a certain variable. While these methods may be useful for a holistic analysis of the data, they all assume that each variable is equally important. However, this is almost never the case, because not all variables in a real-world dataset are created equal. In fact, it is often the case that only a handful of variables have a tangible effect on the outcome (Raval, "Dimensionality Reduction"). In these cases, where a reasonable inference or analysis can be based off of some (and not all) of the variables in the dataset, it is not necessary to provide representation for all variables in a graphical visualization of the dataset. As such, the visualization can be simplified by eliminating insignificant variables, allowing the viewer to hone in on only the most important information in the dataset. The mathematical methods by which variables in a dataset are eliminated (reduced) fall into category of computation called *Dimensionality Reduction*.

2.1 Principal Component Analysis

One common method of dimensionality reduction is Principal Component Analysis (PCA), which aims to "extract the important information from the [dataset], to represent it as a set of new...variables called principal components, and to display the pattern of similarity of the observations and of the variables as points in maps" (Abdi and Williams 1).

That is to say that PCA functions by stretching and rotating the data in such a way that the lower-dimensional representation of the data displays the relative relationships among the data points without altering the data itself. The goal of PCA, therefore, is to display axes of relationships where the variance (spread) of the data is maximized. In this way, the lower-dimensional graph depicts the most important axes while eliminating axes where the data is quite similar ("StatQuest: Principal"). I found that the easiest way to interpret PCA was by thinking of it as selecting the two or three directions where the data is the most varied and mapping those directions onto the axes of a conventional graph. This works because the

[2]For examples of the these visualization techniques, see the Appendix - "Higher-Dimensional Visualization Examples"

directions where the spread of the data is the greatest are bound to be the most important directions for deciding where each data point is placed in higher-dimensional space.

The first step in PCA is to center the data such that

$$\sum_{i}^{N} Z_i = 0,$$

where Z is the matrix of data (Osadchy, "Dimensionality Reduction"). This implies that half of the data lies to the left of the y-axis while the other half lies to the right, essentially ensuring that PCA is able to come to valid conclusions about the data even though the ranges of the some variables might be skewed in the original dataset. In order to center the data, one can simply subtract the mean from each data point for $i \in N$:

$$X_i = Z_i - \hat{\mu},$$

such that

$$\hat{\mu} = \frac{1}{N} \sum_{i}^{N} Z_i$$

(Osadchy, "Dimensionality Reduction").

Once the data has been centered, the *covariance matrix* can be computed. This covariance matrix is the set of vectors corresponding to the directions[3] in which the data is spread. The matrix is computed by calculating the eigendecomposition[4] of the matrix multiplication of $X^T X$ (Raval), which in this case is given by

$$W = \frac{1}{N} \sum_{i=1}^{N} X^T X,$$

where W is the covariance matrix (Osadchy, "Dimensionality Reduction"; Brunton, "Dimensionality Reduction")[5]. Once the matrix has been calculated, the aforementioned directions are sorted from largest to smallest by the amount of spread in each direction. This sorting allows for the arrangement of the covariance vectors by most variance to least (Brunton). In order to create a lower-dimensional representation of the data using PCA, one need only to choose r number of covariance vectors such that $1 \leq r \leq 3$.

This allows for the most important (i.e. most variable) relationships among the data to be displayed by graphing the data on these covariance vectors (directions), which indicate the relative spatial relationships of the data points in those directions where the data is the most spread out (Raval). The location of each data point on a PCA graph is given by

[3]N.B. These directions do not necessarily lie along normal principal axes (i.e. x, y, z) but lie instead on any line in any direction.

[4]An eigendecomposition of a matrix multiplication yields the vectors which do not change direction even after the linear transformation caused by the matrix multiplication. In this case, these vectors (called eigenvectors) indicate the directions of the spread of the data, which form the covariance matrix. A visual explanation of this process can be found at *https://www.youtube.com/watch?v=PFDu9oVAE-g*

[5]N.B. In this investigation, n refers to the number of dimensions in a dataset whereas N refers to the number of data points in a dataset.

$$T_r = W_r X,$$

where T_r is the matrix of data points corresponding to r number of covariance vectors (i.e. how many dimensions will be present in the lower-dimensional visualization) (Brunton). That is to say that

$$T_r = \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_N^1 \\ \vdots & \vdots & \ddots & \vdots \\ y_1^r & y_2^r & \cdots & y_N^r \end{bmatrix}$$

(Osadchy, "Dimensionality Reduction"). In the matrix T_r, each column represents a single data point in the dataset and each row represents a principal component.

Fig. 1 shows an example of PCA used to transform three-dimensional data into a two-dimensional visualization. The result of a PCA graphing is that relationships among data points are visualized through the grouping of data points on the PCA graph ("StatQuest: Principal"). As such, I found that PCA is extremely useful for the classification of data points into multiple categories, as the grouping of data serves to classify data points by group.

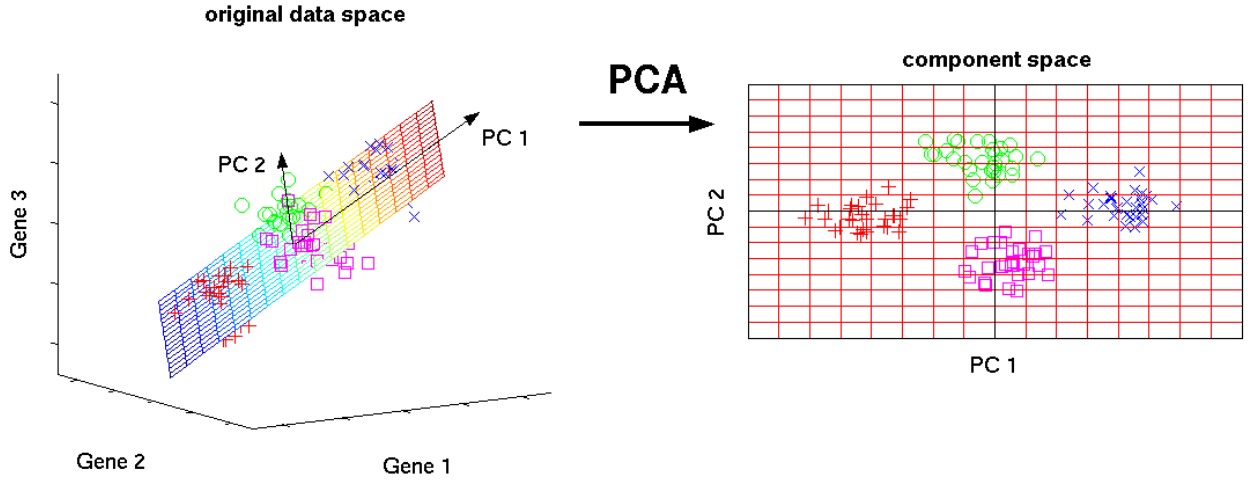

Figure 1: Example of Principal Component Analysis Performed on a Three-Dimensional Data Set (Scholz)

Since the principal components of PCA aren't necessarily principal axes in a normal Cartesian coordinate system (i.e. x, y, z, etc.), when PCA is performed and graphed, these principal components do not retain the same axis labels as the original dataset. Instead, axes on a PCA graph are conventionally labeled PC_q or Dim_q for $q \in r$, as the axes represent the calculated principal components of the dataset (Abdi and Williams 4). Despite losing the original labels for the features (dimensions) in the data, these labels are still effective because their purpose is not to show how data relates to a variable, but rather to show how data relates to other data. As such, the labels do not need to be descriptive as the inference

to be gained from PCA can be gained by looking at the grouping of data points on the PCA graph (Osadchy, "Lecture: Introduction").

By far the hardest part of learning PCA for me was understanding the linear algebra behind why the eigendecomposition of the data yields the principal components. However, when I started to think of the PCA process in a more computational sense by looking at the summation notation for the covariance matrix and examining what each value in the T_r matrix means, I found that PCA was far more intuitive than I had previously thought. All PCA is doing is finding the two or three lines in higher-dimensional space where the data is most spread out and rotating those lines so that they lie on the axes of a two- or three-dimensional graph.

2.2 Multidimensional Scaling

Another dimensionality reduction technique is Multidimensional Scaling (MDS). As with PCA, MDS aims to reduce dimensionality by focusing on preserving the spatial relationships among the data. However, while PCA achieves this by only visualizing a few individual principal components, MDS preserves the spatial relationships among the data across all dimensions (Osadchy, "Dimensionality Reduction"). It achieves this by preserving the pairwise (i.e. focusing on a pair of data points at a time) distances between the nodes (data points) when plotted onto a two- or three-dimensional graph (Jung). In this way, MDS projects the data down to a lower dimension without simply truncating the original dimensionality of the data (which is what occurs with PCA) (Brunton). Whereas PCA concerns itself with the relative locations of each data point, MDS concerns itself not with a location, but with a distance for each pair of data points (Osadchy, "Dimensionality Reduction").

The distance[6] between data point i and data point j is given by

$$d_{ij} = ||x_i - x_j||$$

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + \cdots + (n_i - n_j)^2},$$

which is to say that d_{ij} is the n-dimensional distance between points i and j (for $i, j \in N$ and $i \neq j$) (Jung; Buja et al 2-3). This distance can be calculated using the existing data alone (i.e. by applying the Euclidean distance formula notated above), thus serving as the starting point for MDS.

An MDS visualization is most commonly achieved through metric multidimensional scaling (mMDS), which finds the optimal location for each data point on a graph by minimizing a *stress function*. The stress function provides a measure of how well each of the previously-calculated pairwise distances is being preserved in the visualization (Jung). This stress function is denoted by

[6]N.B. For the purposes of MDS, x_i when used alone denotes the n-dimensional coordinates of data point i and when used along with other variable letters (e.g. $y_i, z_i, \cdots, n_i$) denotes the first dimension of data point i. As such, $||x_i - x_j||$ denotes the magnitude of the difference between the two data points i and j (up to the n^{th} dimension). Conversely, $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ denotes the Euclidean distance between data points i and j strictly in two dimensions (x and y).

$$Stress(x_1, x_2, \cdots, x_N) = \sum_{j=1}^{N-1} \sum_{i=1}^{N-1} (||x_i - x_j||_2 - d_{ij})^2,$$

(Buja et al 3).

In other words, the stress function takes as inputs each data point and outputs the cumulative error between the desired two-dimensional distances ($||x_i - x_j||_2$), which correspond to the distances between the points on the MDS graph, and the actual n-dimensional distances (d_{ij}) for each combination of points i and j (again where $i, j \in N$ and $i \neq j$) (Jung). Using this metric, the exact two-dimensional coordinates for each point's location in the MDS graph can be extracted by optimizing the stress function to decrease the 'strain' on the distances, which can be achieved through a non-linear optimization algorithm (Hart).

A common example of mMDS is action is using pairwise distances (airplane mileage is commonly used) between U.S. cities to develop a two-dimensional map of the location of the cities. In *Fig. 2*, mMDS correctly plots the relative locations of major U.S. cities given their distances to each other.

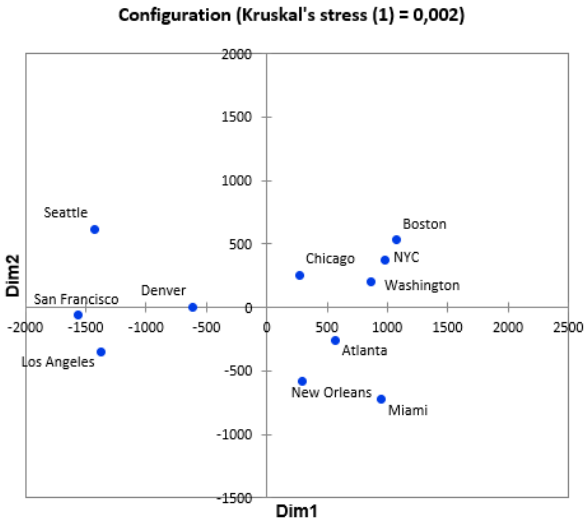

Figure 2: Example of Multidimensional Scaling to Locate U.S. Cities Based on their Pairwise Distances ("Multidimensional Scaling")

An MDS visualization is, much like PCA, excellent at recognizing groups among various dimensions in a dataset. The problem I see with MDS is that by classifying the data points, the algorithm eliminates the precision and measurability that goes along with using labels for each variable. Here too, the axis labels do not take on any specific measurement, but are constructed simply for the purpose of the visualization. Thus, the grouping comes at the cost of being unable to intuitively quantify data points according to their variables.

3 Non-Linear Dimensionality Reduction

Both PCA and MDS fall under the category of *linear* dimensionality reduction because their methods of calculating variance and distance of data points, respectively, rely on connecting two (or more) data points using an application of Euclidean (linear) distance. However, linear relationships aren't always the best indication of the relationships among data points. This is the case for datasets where the data creates a multidimensional *manifold* (Ghodsi). A manifold is a topology (a shape or form) that is "locally Euclidean" (Rowland). This means that when the manifold is zoomed in to a very close distance, the points that comprise the manifold behave in an essentially continuous fashion, where neighboring points on the manifold are nearly identical to the original point (Ghodsi). Data that manifests itself as a manifold is ill-suited for linear dimensionality reduction because the only way to get from one point on the manifold to another is to 'walk' along the surface of the manifold. In that way, drawing a straight line from the first point to the desired point takes a shortcut that disobeys the essential property of a manifold (Osadchy, "Lecture: Manifold"). As such, non-linear dimensionality reduction techniques are required to preserve the properties of the original manifold when cast down to two (or three) dimensions (Relstab).

3.1 Isomaps

An Isomap is a lower-dimensional visualization much like MDS in that it preserves the pairwise distances between all points in the data set while mapping them onto a two- or three-dimensional graph. However, instead of maintaining the pairwise Euclidean distances, Isomap maintains the pairwise geodesic distances – the distance between two points along the surface of a topology – in the higher-dimensional manifold (Yang et al). Whereas for the three-dimensional manifold seen in *Fig. 3*, MDS would compute the blue Euclidean distance between x_1 and x_2, an Isomap algorithm would compute the red geodesic distance between x_1 and x_2 (Osadchy, "Lecture: Manifold").

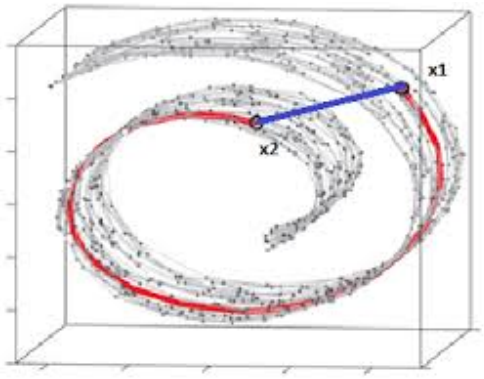

Figure 3: Example of How Geodesic Distance Provides a More Accurate Distance Measurement on Higher-Dimensional Manifolds (Alberti)

Because the Swiss roll depicted in *Fig. 3* has a quasi-continuity among its points that

forms its distinctive shape, using MDS would eliminate that shape and thereby, eliminate a primary characteristic of the data when projected down to two dimensions. The solution that retains the essential shape of the manifold is to use an Isomap, which creates a lower-dimensional graph that maintains the continuity of the higher-dimensional manifold (as seen in *Fig. 4*) (Tibshirani).

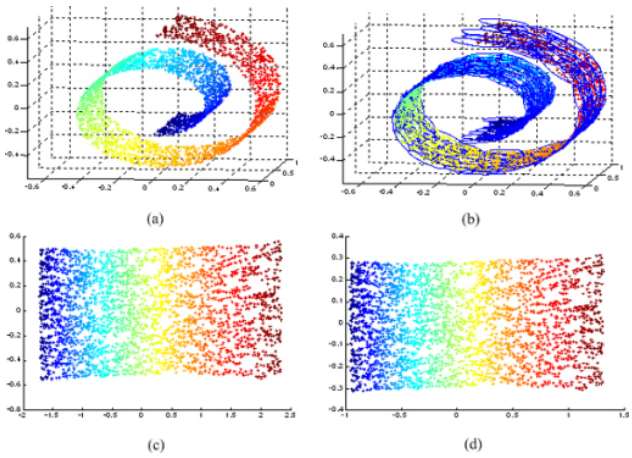

Figure 4: Example of Isomap Non-Linear Dimensionality Reduction Method (Lei et al)

An Isomap is created in three steps. The first is to convert the manifold to an *adjacency graph*, where the vertices of the graph represent data points and edges connect neighboring points (Ghodsi). This adjacency graph is an approximation of the original manifold and can be calculated either by running a *K-Nearest Neighbors Algorithm* (KNN), which finds k number of points adjacent to each point on the manifold, or by connecting to all points within a certain radius (as was done in *Fig. 4* step (b)) (Osadchy, "Lecture: Manifold").

The second step is to estimate the shortest geodesic distance that connects each pair of points on the adjacency map. This is can be computed using Djikstra's shortest path algorithm, which traverses the adjacency graph until it comes across the shortest path through the edges of the graph (Osadchy, "Lecture: Manifold"). This shortest geodesic distance d_{ij}^G between points i and j (for $i, j \in N$ and $i \neq j$) is the Isomap equivalent of d_{ij} for MDS, in that it is the measure of the higher-dimensional distance of the data (Tibshirani).

Finally, mMDS is performed on the geodesic distances such that

$$Stress(x_1, x_2, \cdots, x_N) = \sum_{j=1}^{N-1} \sum_{i=1}^{N-1} (||x_i - x_j||_2 - d_{ij}^G)^2,$$

where

$$||x_i - x_j||_2 = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

(Ghodsi).

The result, when the stress function is minimized, is a set of two-dimensional coordinates for each data point in the data set that preserves the geodesic distances between points when

plotted on a two-dimensional graph (Tibshirani). The entire Isomap process maps the Swiss roll in the example above (*Fig. 4* step (a)) to the flat representation in step (c), which maintains the color continuity of the original manifold (Ghodsi).

Of course, understanding Isomap first requires one to understand MDS, which is why there is often a great barrier to using non-linear dimensionality reduction for persons who don't have a good grasp of linear dimensionality reduction. For me, however, I found Isomap to be extremely intuitive, as its process can be characterized in the above example as the process of unrolling the Swiss roll to form a flat surface. In this way, I like to think about Isomap as trying to ensure that the higher-dimensional manifold doesn't get too stretched out or torn apart when you try to 'unroll' it so it can be shown in a lower dimension.

3.2 Locally Linear Embedding

Locally Linear Embedding (LLE) is another non-linear dimensionality technique that, unlike Isomap, tackles the problem of manifold dimensionality reduction from a local, and not global standpoint (Osadchy, "Lecture: Manifold"). As the name suggests, LLE assumes that the higher-dimensional manifold can be approximately characterized by the collection of linear patches tangent to the surface of the manifold on a local (very small) level (*Fig. 5*) (Relstab). Applying this assumption allows you to treat points in the immediate neighborhood (vecinity) of a certain point on the manifold as lying on the same plane as that point, making it easier to transform the manifold down to lower dimensions (Ghodsi).

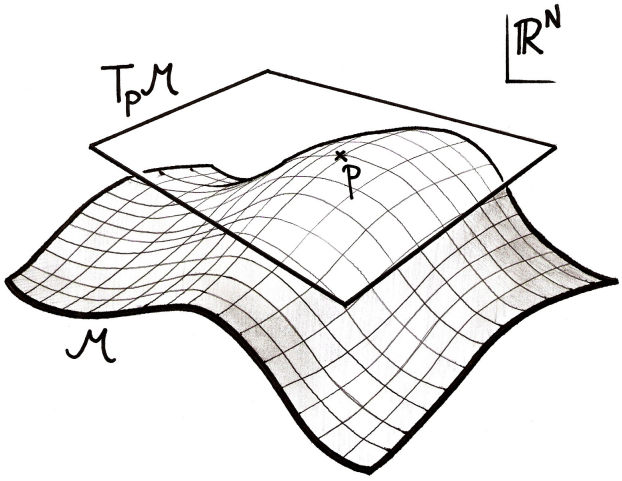

Figure 5: Three-Dimensional Manifold with Planar Patch T_pM Tangent at Point p (Oblak)

The first step in LLE, as with Isomap, is to develop an adjacency graph of the data points in the dataset by using the K-Nearest Neighbors (KNN) algorithm (Osadchy, "Lecture: Manifold"). LLE then aims to calculate weights for the lines that connect every vertex (data point) in the adjacency graph. It does this by minimizing a *reconstruction error* for the reconstruction of the location of vertex x_i given the locations of the neighboring vertices. The reconstruction error minimization function is given by

$$\varepsilon(W) = \sum_{i=1}^{N} ||x_i - \sum_{j=1}^{k} W_{ij} x_j||^2,$$

(Ghodsi).

Thus, this weight optimization function finds a matrix W of weights representing every connection in the adjacency graph such that the total distance between the current node x_i is as close to the sum of each neighboring node's position multiplied by the weight W_{ij} from the current node x_i to the neighboring node x_j for $j \in k$, where k is a user-defined parameter for the number of neighbors found by the KNN algorithm (Osadchy, "Lecture: Manifold"). That is to say that $\varepsilon(W)$ aims to create a set of weights so that each point on the original adjacency graph can be reproduced only by knowing the locations of its neighbors.

Finally, using the weights calculated in the above step, the LLE algorithm performs a lower-dimensional embedding similar in process to MDS. Namely, LLE optimizes the location of each node on the lower-dimensional representation so that the *stress* function $\phi(y)$ given below is minimized.

$$\phi(y) = \sum_{i=1}^{N} ||y_i - \sum_{j=1}^{N} W_{ij} y_j||^2,$$

where $i \neq j$ (Ghodsi). The optimization of the above function yields two-dimensional coordinates (the matrix y) for every data point in the dataset so that the reconstruction of the manifold (in two dimensions) using the coordinates of neighboring points and their associated weights results in the least strain on the stress function (Osadchy, "Lecture: Manifold"). *Fig. 6* shows how LLE considers small neighborhoods of points at a time (the regions enclosed by the black outlines in steps B and C) in order to recreate the manifold in two dimensions .

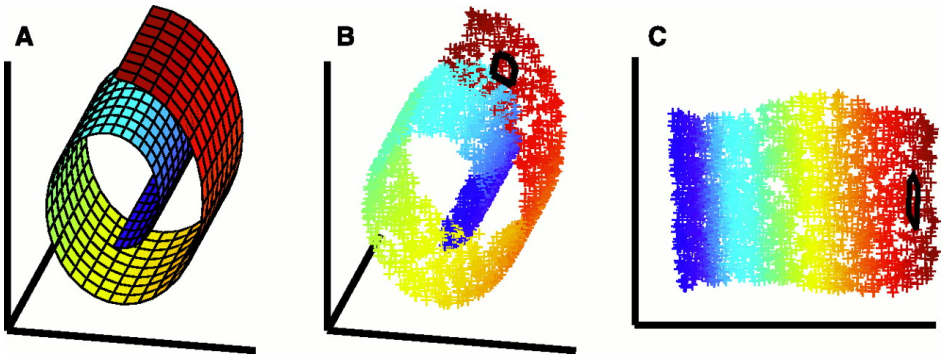

Figure 6: Example of Locally Linear Embedding that Preserves Pairwise Relationships on a Local Level (Roweis and Saul)

I find that the best way to think of the difference between Isomap and LLE is to always keep in mind that, whereas Isomap tries to reconstruct the manifold all at once (i.e. on a global level), LLE approaches the reconstruction process step-by-step in that is focuses on

small (local) regions of the manifold at a time (Osadchy, "Lecture: Manifold"). Because LLE doesn't try to come up with a set of lower-dimensional points that preserve geodesic distance all in one large optimization process, LLE is a significantly faster algorithm as the multiple smaller optimization problems take less time than Isomap's enormous optimization puzzle (N^2 algorithmic time complexity versus N^3 time complexity, where N is the number of data points in the dataset) (Ettinger).

Part IV
Evaluation

4 Conclusions

It should also be noted that the dimensionality reduction techniques detailed above in no way provide an exhaustive list of processes for the visualization of higher-dimensional data. Both *T-Distributed Stochastic Neighbor Embedding* (t-SNE) and *Laplacian Eigenmaps* are linear and non-linear methods of dimensionality reduction, respectively, that produce equally insightful visualizations as the aforementioned methods.

5 Reflection

Overall, I really enjoyed exploring dimensionality reduction from a mathematical perspective as opposed to stumbling upon the methods as somewhat of a buzzword featured as a tool on a data processing application. This is because I believe that having a thorough, intuitive understanding of dimensionality reduction and higher-dimensional visualization will allow me to seamlessly apply these techniques to a variety of data science and computer science applications I come across in the future.

To that end, I was surprised to see just how prevalent these techniques are in everyday data science. Particularly, the applications of both PCA and MDS were simultaneous really intuitive and not immediately obvious in the sense that the application of dimensionality reduction makes sense in those situations, but I would never have had thought that I should use dimensionality reduction in those scenarios. This holds especially true for MDS, which has a variety of fascinating real-world examples, the primary of which is the aforementioned example of mapping U.S. cities based on their distances to each other. At first, this seemed to be a quite simple and straightforward task to do, seeing as we can already map the cities by taking a picture of the U.S. or by measuring the distances and directions from a fixed location (such that a map can be constructed by placing each city at the tip of the vector extending from that fixed central location). On second thought however, it occurred to me that MDS might be an extreme beneficial tool for mapping objects where their actual locations are harder to actually measure or determine. One such example is the position of satellites in space; if the time taken for a radio wave to be transmitted from one satellite to the next can be calculated, a set of relative distances between satellites can be used to craft a map of the location of each satellite in space using MDS. Another example of a real-life

application of MDS is for use in the user interface design for mobile applications. Given data about what each button or feature in a mobile application is supposed to do, MDS can decide where each button or feature should be placed on the screen to minimize travel time between two buttons or features with similar functions. This example in particular intrigued me, as I am very interested in user experience and user interface design as it related to digital applications and programs.

Most important, I appreciated the concrete applications dimensionality reduction can and will have in my work on machine learning.

I believe that dimensionality reduction

I am nevertheless conscious of the amount of prerequisite knowledge required in order to fully understand these dimensionality reduction techniques. In fact, while researching these techniques, I often found myself needing to step back and take a break in order to parse through the tremendous amount of information I had been presented with. More than a couple of times, this involved going back to the basics of graphing or linear algebra, writing out concrete examples of how each dimensionality reduction technique worked, or watching online videos to give myself a break from reading strictly academic papers. However, doing so *always* enabled me to finally understand the mathematical theory involved, and to explain concepts to myself (which I find to be immensely important for my own understanding of mathematical concepts and principles). As such, my recommendation to anyone trying to learn about dimensionality reduction for the first time would be to do it in steps, always ensuring you understand previous concepts before moving on to more difficult ones. Additionally, reading (or watching) multiple sources on this topic will allow you to understand complex concepts from a variety of perspectives (e.g. a purely mathematical perspective, a real-life data science perspective, a computational perspective, etc.), ultimately helping you to find the perspective that helps you to understand the subject matter in the most intuitive manner. After all, this paper is only *my* attempt to explain the subject of dimensionality reduction from my perspective as a budding computer scientist and data scientist.

Part V
References

Abdi, Herv, and Lynne J. Williams. "Principal Component Analysis." *Wiley Interdisciplinary Reviews: Computational Statistics*, 2010, www.semanticscholar.org/paper/Principal-component-analysis-Abdi-Williams/4fdbd442a8b5d74cc3c8c3437cf4acd68c2ee60c. Accessed 31 May 2018.

Alberti, Matteo. "Euclidean Distance Vs. Geodesic Distance on Swiss Roll Manifold." 27 Nov. 2017. *Deep Learning Italia*, DeepLearningItalia, 27 Nov. 2017, www.deeplearningitalia.com/manifold-based-tools-isomap-algorithm/. Accessed 30 May 2018.

Buja, Andreas, et al. *Data Visualization with Multidimensional Scaling*. Yale U Department of Statistics and Data Science, 18 Sept. 2007. *Yale Department of Statistics and Data Science*, www.stat.yale.edu/ lc436/papers/JCGS-mds.pdf. Accessed 30 May 2018.

"Dimensionality Reduction: Principal Components Analysis, Part 1." *YouTube*, uploaded by Bing Brunton, Google, 10 June 2016, www.youtube.com/watch?v=ZqXnPcyIAL8. Accessed 30 May 2018.

"Dimensionality Reduction - the Math of Intelligence #5." *YouTube*, uploaded by Siraj Ravel, Google, 14 July 2017, www.youtube.com/watch?v=jPmV3j1dAv4. Accessed 30 May 2018.

Ettinger, Evan. "Global (Isomap) Versus Local (LLE) Methods in Nonlinear Dimensionality Reduction." U of California San Diego. 12 May 2005. Presentation slides.

Ghodsi, Ali. "Lecture 4: MDS, Isomap, LLE." 17 Jan. 2017. *YouTube*, Google, 23 Jan. 2017, www.youtube.com/watch?v=RPjPLlGefzw. Accessed 30 May 2018. Lecture.

Grinstein, Georges, et al. *High-Dimensional Visualizations*. Institute for Visualization and Perception Research, U of Massachusetts Lowell, 2002. *Semantic Scholar*, pdfs.semanticscholar.org/43f7/66c06e2a7770d9f37dcd9cfff5bd5dcfc22f.pdf. Accessed 23 May 2018.

Hart, John C. "Data Visualization: Multidimensional Scaling." 2016. *YouTube*, Google, www.youtube.com/watch?v=qZ_QHXJ29iw. Accessed 30 May 2018.

Jung, Sungkyu. "Multidimensional Scaling." Advanced Applied Multivariate Analysis, U of Pittsburgh. 2013. Lecture slides.

Lei, Y.K., et al. "Swiss Roll Isomap." 2012. *National Institutes of Health*, U.S. National Library of Medicine, 2012, openi.nlm.nih.gov/detailedresult.php?img=PMC3348017_1471-2105-13-S7-S3-13&req=4.

"Multidimensional Scaling Example - US Cities." *XLStat*, Addinsoft, 2017, www.xlstat.com/en/solutions/features/multidimensional-scaling-mds. Accessed 30 May 2018.

Oblak, Blagoje. "Linear Approximation of Manifold." *Inspire*, inspirehep.net/record/1386700/plots. Accessed 30 May 2018.

Osadchy, Rita. "Lecture: Introduction." Unsupervised Learning 2011, U of Haifa. 2011.

Lecture notes.
Osadchy, Rita. "Lecture: Manifold Learning." Unsupervised Learning 2011, U of Haifa. 2011. Lecture notes.
Osadchy, Rita. "Dimensionality Reduction: PCA, MDS." Unsupervised Learning 2011, U of Haifa. 2011. Lecture notes.
Relstab, Andrew. "Locally Linear Embedding." 13 Mar 2017. *YouTube*, Google, https://www.youtube.com/watch?v=scMntW3s-Wk. Accessed 30 May 2018.
Roweis, Sam T., and Lawrence K. Saul. "Locally Linear Embedding Local Approach." *Science Magazine*, American Association for the Advancement of Science, 22 Dec. 2000, science.sciencemag.org/content/290/5500/2323.full. Accessed 4 June 2018.
Rowland, Todd. "Manifold." *MathWorld*, Wolfram Research, mathworld.wolfram.com/Manifold.html. Accessed 30 May 2018.
Scholz, Matthias. "Graph of Principal Component Analysis." *Nonlinear PCA*, www.nlpca.org/. Accessed 30 May 2018.
"StatQuest: Principal Component Analysis (PCA) Clearly Explained (2015)." *YouTube*, uploaded by Josh Starmer, Google, 13 Aug. 2015, www.youtube.com/watch?v=_UVHneBUBW0. Accessed 30 May 2018.
Tibshirani, Robert. "Non-Linear Dimension Reduction." STA306B - Unsupervised Learning, Stanford U. 23 May 2011. Lecture slides.
Yang, Bo, et al. "Multi-Manifold Discriminant Isomap for Visualization and Classification." *Pattern Recognition*, vol. 55, July 2016, pp. 215-30. ScienceDirect, www.sciencedirect.com/science/article/pii/S0031320316000571. Accessed 30 May 2018.

Part VI
Appendix

5.1 Higher-Dimensional Visualization Examples

5.1.1 Radar Chart

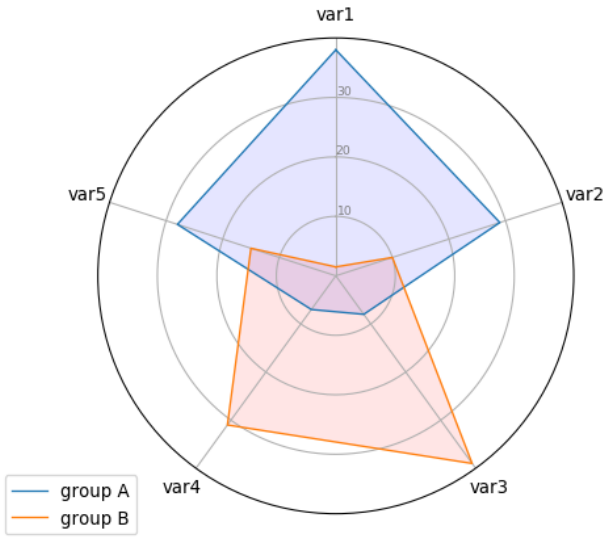

Figure 7: Radar Chart of Two Data Points in Five Dimensions

5.1.2 Scatter Plot Matrix

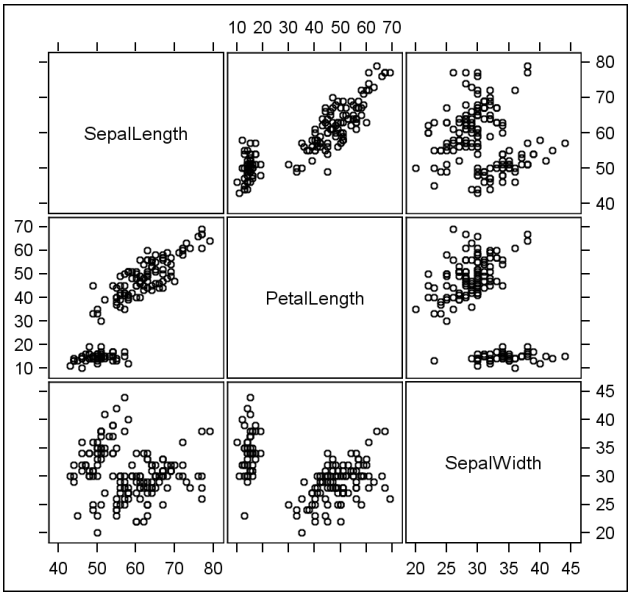

Figure 8: Scatter Plot Matrix with Three Dimensions

5.1.3 Parallel Coordinate System

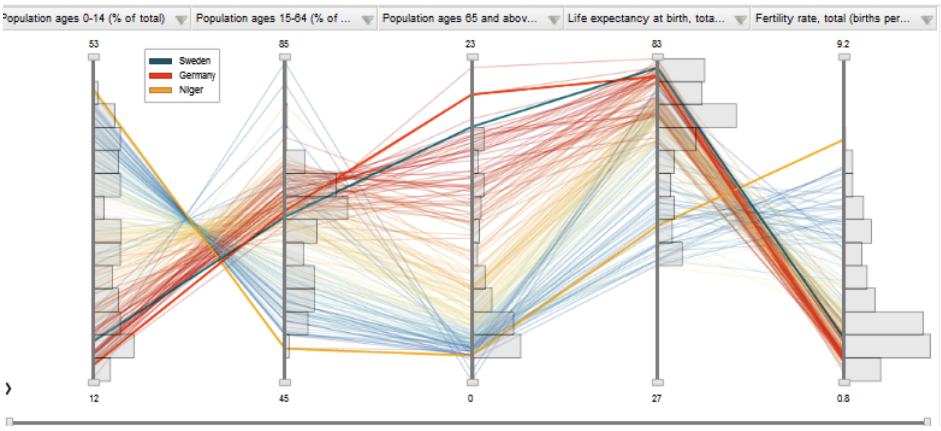

Figure 9: Parallel Coordinates Graph with Five Dimensions