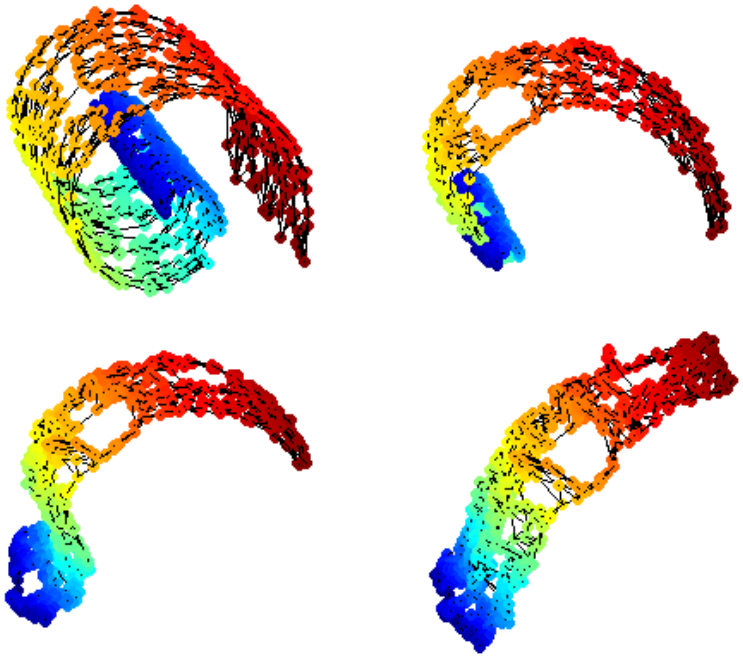
Manifold Learning and Diffusion Maps



Mathematical and Computational Foundations of Data Science Spring 2023

16 April 2023

Anshika Agrawal, Bryan Munoz, Nubaira Milki, Krutal Patel

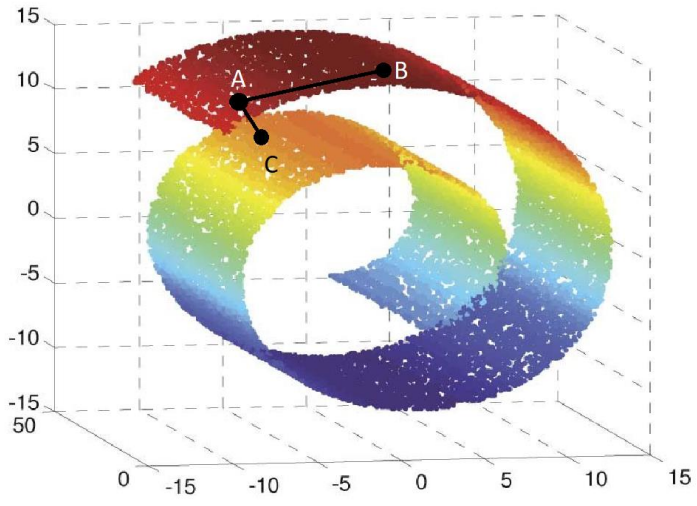
A. Introduction

Data sets oftentimes come with high amounts of features and entries. The number of entries is correlated with the reliability, validity, and reproducibility of models derived from the data. However, having more features can burden the algorithm and increase processing time. Removing features may also erase key information. It is crucial to be able to remove features while still maximizing the amount of information kept in the data. Manifold learning is a machine learning algorithm used for exactly this purpose.

Manifolds are the topological surface of a shape. They are not necessarily planar, but rather homeomorphic to an open subset of Euclidean space. The underlying philosophy of manifold learning is that data points come from a low-dimensional manifold that is embedded in a high-dimensional space. Manifold learning uncovers this low-dimensional representation of the data as well as its parameters. [3]

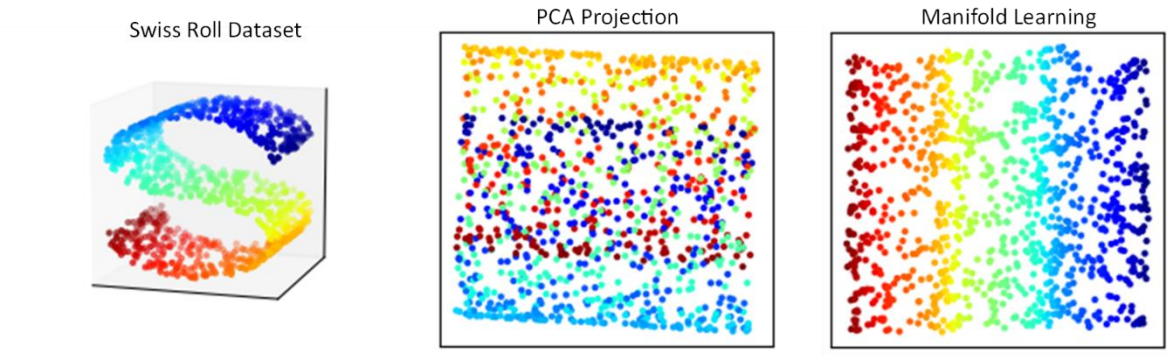
Manifold Learning is particularly effective with nonlinear, high-dimensional data, unlike Principal Component Analysis (PCA). PCA is a linear dimension reductionality technique that aims to extract the direction of greatest variance from given data. PCA preserves the relative distances between data points in both the higher and lower dimension spaces. Particularly, data points closer to each other in high dimensional space are often mapped close together in the lower, transformed dimension space. This can raise some issues as high dimensional data isn't always linear or uniform in nature.

For example, consider the following standard *Swiss roll* dataset in Figure 1.1:



Consider the data points A and B on the *Swiss roll* and notice how both data points belong to the same class, marked by the same color. If we now consider data point C, we can note the distance $\big||AB|\big| > \big||AC|\big|$, despite data point C having another class label. If PCA were to be applied to this dataset, the distances will be preserved, and data points A and C will be noted to be more" similar" than data points A and B. This is a clear misrepresentation of the high dimensional data since A and B belong to the same class. Manifold learning methods often consider such data points and relationships, which is why they are often preferred over PCA when analyzing large scale nonlinear data in higher dimensions.

For example, consider the following set of visualizations produced by two Manifold learning techniques (t-SNE and Locally Linear Embedding) and Principal Component Analysis from the *Swiss Roll* Dataset.



The effect of the nonlinearity in the dataset produces significant differences in the 2-dimensional projections of the original dataset.

B. Laplacian Eigenmaps

B1. Introduction

Laplacian Eigenmaps is an unsupervised technique used to perform dimensionality reduction on a data set that cannot be linearly separated. The main idea behind this technique is to transform the data into a weighted graph where you can then solve for a matrix unique to the graph called the Laplacian. The Laplacian conserved locality between neighbors allowing for a reduced system to be solved for [5].

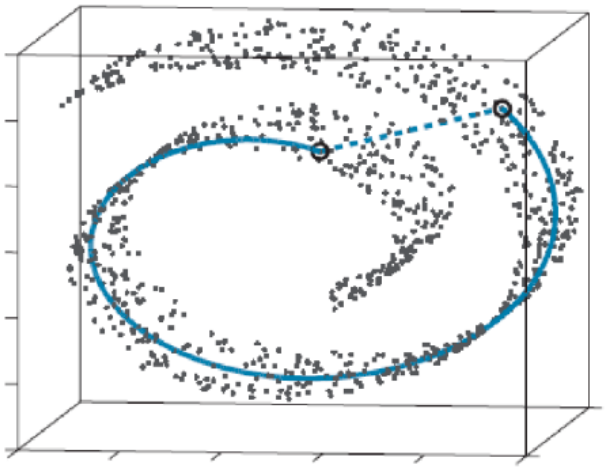
B2. Theory

Stage 1 – Converting the Data Set into a Graph:

In converting the data set into a graph, the vertices are set to be the data points and the edges must be arbitrarily determined.

There are two main strategies for determining the edges between points: N nearest neighbors and ε-neighborhood [16].

N-nearest neighbors involve finding the distance, specific to the manifold, between a vertex and other vertices and then choosing the N neighbors whose distance is smallest. The benefit of using this strategy is that it will always produce a connected graph [16]. An important factor to note is that if the manifold is a curved surface, then it is important to use the geodesic distance since using the Euclidean distance does not conserve the locality between the vertices. An example of this is the Swiss Roll where the geodesic distance between vertices must be used to calculate the distance [8]. Another difficulty with choosing this method is that it is more



computationally expensive than ε-neighborhood because all the distances between the vertices must be calculated and then compared to determine which distances are the smallest.

ε-neighborhood determines a cutoff value for distances to be so that an edge may be drawn between two vertices. This method is more computationally efficient than n nearest neighbor, but it can lead to a disconnected graph being made [8]. Having a disconnected graph makes it harder to solve for the reduced dimensions.

The next step is to determine the weights of the edges. There are two main strategies for determining the weight of each edge: Heat Kernel and Simple Weights [5].

The equation for the heat kernel is given as follows:

$$e^{\frac{\left|\left|x_i - x_j\right|\right|_2}{t}}$$

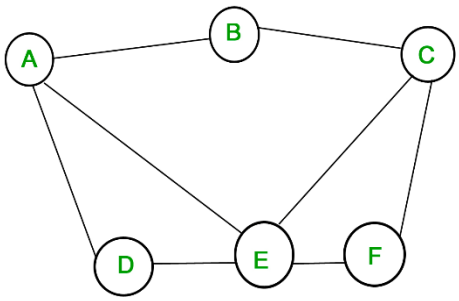
Where *t* is the parameter value that must be specified. This leads to the difficulty of deciding which *t* value to use. There is no correct way to determine what value *t* should be so it must be fined-tuned specifically to each data set.

The simple weights method makes the value of a weighted edge one if it connects two vertices and zero if the vertices are not connected. This process can be computed more quickly but does not allow for more precision when assigning locality [16].

Stage 2 – Determining the Laplacian of the Graph:

To solve for the Laplacian, we must first solve for the adjacency matrix and diagonal degree matrix [5].

The adjacency matrix is a square matrix describing the edges between a vertex and every other vertex, which means it is a symmetric matrix. An example of this is solved below [10].



$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Where the first column and row correspond to vertex A, second column and row is B, etc.

Equation for each diagonal on the degree matrix is

$$D_j = \sum_{i=0}^{N-1} W_{ij}$$

The degree of a vertex, which is the value in the diagonal of the matrix, is the sum of the weights on that vertex [8]. An example of the degree matrix is calculated below using the graph shown above as an example.

$$\begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Before solving for the Laplacian, it is important to define it as a concept. The graph Laplacian is the flux density of the gradient flow of a graph but can also be thought of as more simply as the matrix representation of the graph [11]. From here solving for the Laplacian is very easy since the equation is just:

$$L = D - W$$

Since the Laplacian is a matrix representation of the graph, we can solve for the eigenvectors to find the basis on which the dataset exists on. It is very important to note that the eigenvalues are ordered from least to greatest in this context [16]. If the graph is a connected graph, the first eigenvalue will be zero since this corresponds to the eigenvector filled with ones since it points in the direction of the graph. The more separate pieces of the graph, the more corresponding eigenvalues will be zero [8]. This is why it is easier to create a connected graph. To solve for the k dimensional reduced space the first k eigenvectors, corresponding to nonzero eigenvalues.

This last step is very related to Spectral Clustering since once the dimensions have been reduced to a linearly separable basis, it is then very easy to do simple clustering algorithms like K-means to then cluster the data [11].

B3. Algorithm

Toolbox Implementation

The *scikit-learn* provides a variety of parameters that allow for fine-tuning of the Laplacian Eigenmaps algorithm, from the manifold package with the function name Spectral Embedding.

Some of the key parameters in the *scikit-learn* implementation of IsoMap include:

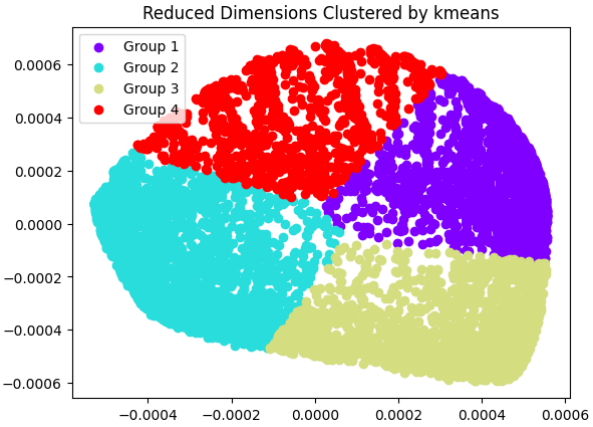
1. n_neighbors: This specifies the number of neighbors that will be used to each data point to construct the neighborhood structure. This is then used to determine geodesic distances between data points. The default is n_neighbors = 5.
2. affinity: Describes technique used to obtain neighbors. The default is nearest_neighbors
3. gamma: determines value of t in the heat kernel, where the default value is 1/len(data_set)
4. n_components: The number of dimensions in the reduced space. The default is n_components = 2.
5. eigen_solver: This parameter determines the method to solve eigenvalue of the Laplacian
6. tol: Specifies the tolerance of the convergence of the eigenvalue solver. The default is tol = 0.

B4. Application

To show the application of Laplacian Eigenmaps and connection to spectral clustering we can reduce the Credit Score dataset and cluster it that can be found in the GitHub repository.

Example 1: Credit Card Dataset

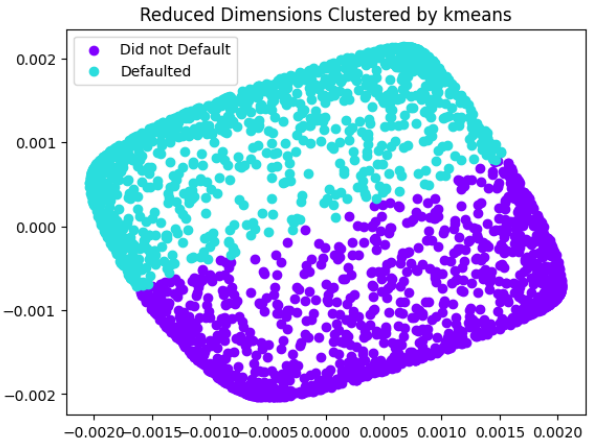
The Credit Card Dataset consists of 17 different features, some of which are Balance, Balance Frequency, Purchases, etc. As well, it contains 8590 rows of data for each column. Using Laplacian Eigenmaps we reduce these dimensions to two and then use kmeans clustering to cluster the data. As shown by the picture below no clear clustering groups are present from reducing the image so kmeans even divides up the image.

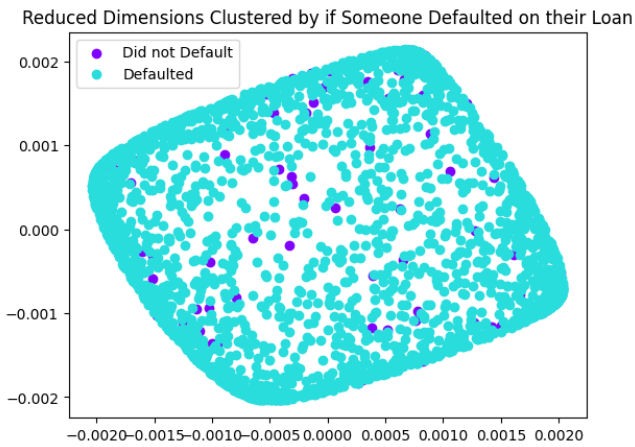


Example 2: Credit Card Dataset

The next dataset consists of 67 different features some of which are graduation status, number of children, if they defaulted on their loan, etc. The dataset tries to see if it is possible to predict whether an individual would default on their loan. However, as shown below no clear clustering groups are shown and even labeling the data (as shown by the picture on the right), the data is

not properly clustered, and it would be difficult to cluster the two groups. This suggests that it would be difficult to predict whether a person would default on their loan or not based on the current data available.





C. Isometric Feature Mapping

C1. Introduction

IsoMap, Isometric Mapping, is an unsupervised [15] dimensionality reduction method that maintains 'geodesic distances' between points that was first proposed in 2000 by Joshua B. Tenenbaum, Vin de Silva, and John C. Langford [16, 18]. Iso-mapping is one of the earliest approaches to manifold learning and is often characterized as an extension of multi-dimensional scaling or kernel PCA [12]. Instead of measuring pure Euclidian distances of data points IsoMap uses geodesic distances which generalize curved surfaces distance and optimizes those distances along the manifold [18]. The general IsoMap algorithm is derived from multidimensional scaling (MDS) but reformed for non-linear dimensionality reduction [17].

C2. Theory

Stage 1 – Nearest Neighbor Search: This step is used to find neighbors on the manifold (*M*) of the dataset based on the distances between the points *i,j* in the space. These points are connected to each other with some fixed radius ε. This data is then presented in a weighted graph *G* over the data points with the individual edges of the weights $d_X(i,j)$ between neighboring points [16].

Stage 2 – Shortest Path Graph Search: IsoMap now estimates geodesic distances between pairs of points on the manifold $d_M(i,j)$ by computing the shorted path distance $d_G(i,j)$ on the weighted graph *G* [12, 16].

Stage 3 – *d*-Dimensional Embedding: This step then applies traditional multidimensional scaling (MDS) to a matrix of distances $D_G = \{d_G(i, j)\}$ which constructs a *d*-dimensional Euclidian

space with the data being embedded within [7]. For the points within this matrix, coordinate vectors y_i are chosen to minimize cost function with D_y being a matrix of Euclidian distances $\left\{d_Y(i,\, j) \,=\, \left|\left|y_i \,-\, y_j\right|\right|\right\}$ and $\left|\left|A\right|\right|_{L^2}$ which is the L^2 matrix norm $\sqrt{\left(\sum_{i\,,j} A_{ij}^2\right)}$. The τ is then used to convert the individual distances calculated into inner products which are used to characterize the geometry of the data. This inner product is defined as $\left|\left|x \,-\, y\right|\right|^2 \,=\, <x,\, x> \,-\, 2<x,\, y> \,+\, <y, y>$ or more generalized for the entire matrix: $D_{ij} \,=\, A_{ii} \,-\, 2A_{ij} \,+\, A_{jj}$ where $D \,\in\, \mathbb{R}^{m\times m}$ [3]. The embedding is created form the positive define matrix inner products D . Then, $D \,=\, \sum_{i\,=\,1}^{n} \lambda_i\, \phi_i \phi_i^T$ and for any $x \,\in \{1,\, \ldots,\, n\}$, we have $\psi(x) \,=\, \left(\sqrt{\lambda_1}\phi_i(x),\, \ldots,\, \sqrt{\lambda_k}\phi_k(x)\right) \in \mathbb{R}^k$ [16].

Advantages and Disadvantages of IsoMap

IsoMap is an increased complexity form of multidimensional scaling (MDS) allowing it to maintain non-linear relationships while MDS is only capable of preserving the linear relationships in a data structure. Several advantages of IsoMap exist including global optimization, it is capable of processing high dimensional data including nonlinear manifolds, IsoMap can optimize the low-dimensions representation globally no matter the input space folding or distortion, and it can guarantee gradual recovery to the real dimension [17].

One of the major disadvantages of IsoMap, however, is that it may be unstable and dependent on the topological space of the data [17]. Small errors in the topology of a dataset can lead to amplified errors within the solution [1]. Choosing the right sized neighborhood in IsoMap to ensure that the dataset is not overly fragmented requires significant optimization within the algorithm which is capable of exasperating any errors present within the data set. Another major disadvantage of IsoMap is for small N, the geodesic distances will be very imprecise when attempting to recover the geometry of the non-linear manifold. This implies that IsoMap should be carefully chosen and primarily applied to larger/higher dimensional datasets as attempting to use it on datasets with low N can create great imprecision [17].

C3. Algorithm

Toolbox Implementation

The s*cikit-learn* provides a variety of parameters that allow for fine-tuning of the IsoMap algorithm [12].

Some of the key parameters in the *scikit-learn* implementation of IsoMap include:

- n_neighbors: This specifies the number of neighbors that will be used to each data point to construct the neighborhood structure. This is then used to determine geodesic distances between data points. The default is n_neighbors = 5.
- n_components: The number of dimensions in the reduced space. The default is n_components = :2.

- eigen_solver: This parameter determines the method to solve eigenvalue solving for geodesic distances.
- tol: Specifies the tolerance of the convergence of the eigenvalue solver. The default is tol = 0.

Complexity and Cost of Function

Stage 1 – Nearest Neighbor Search: Using BallTree for a neighbor search in the data set. The cost of this is $O[D \log(k)\, N \log(N)]$ for *k* nearest neighbors of *N* points in *D* dimensions.

Stage 2 – Shortest-path Graph Search: While the most efficient method is *Dijkstra's Algorithm* which has the cost $O[N^2(k + \log(N))]$, *Floyd-Warshall algorithm* is another efficient method with the cost $O[N^3]$. These can be specified in IsoMap using "path_method" and if not specified the code will choose the best option.

Stage 3 – *d*-Dimensional Embedding: The IsoMap embedding is lastly encoded into the eigenvectors corresponding to the *d* largest eigenvalues of the N x N IsoMap kernel. The cost of this decomposition is approximately$O[dN^2]$ for dense data and can be improved using ARPACK solver. The specific solver that needs to be used can be specified using "eigen_solver" and if not specified the code will choose the best option.

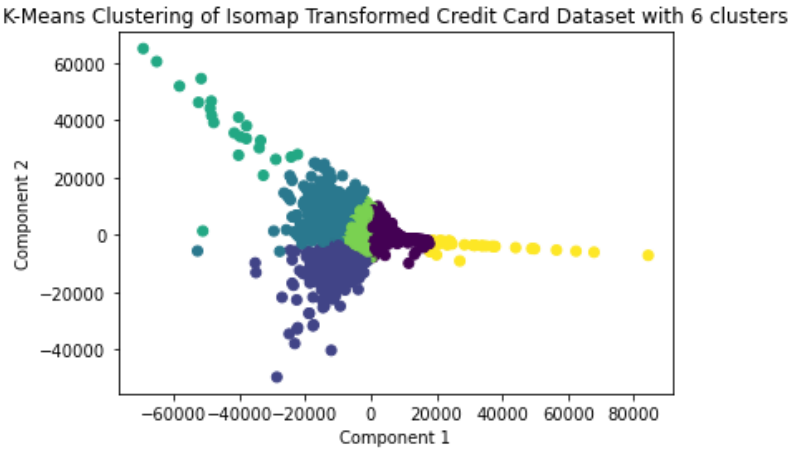
Overall IsoMap complexity: $O[D \log(k)\, N \log(N)] + O[N^2(k + \log(N))] + O[dN^2]$

Where: *N* – number of training data points, *D* – input dimension, *k* – number of nearest neighbors, and *d* – output dimension [12].

C4. Application

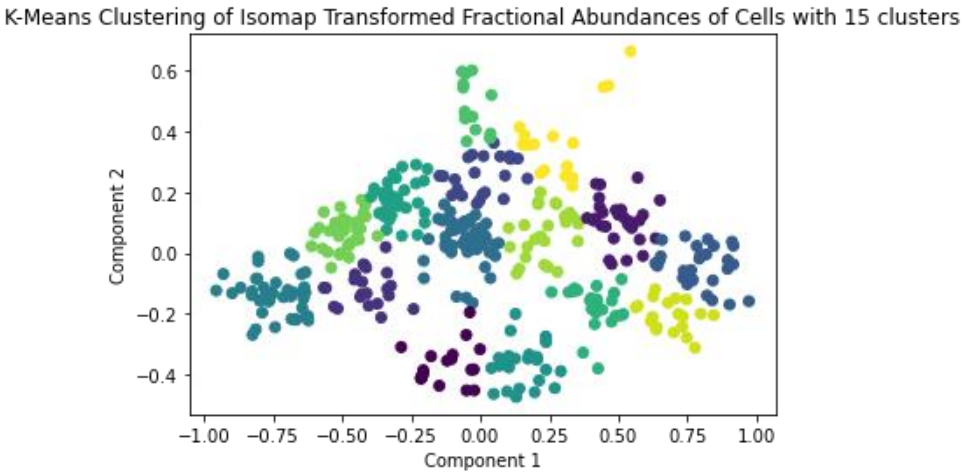
Example 1: Credit Card Dataset

The credit card dataset summarizes the past 6 months of usage behaviors for 9000 active credit card holders. There are several different attributes to characterize credit card usage including Balance, Balance Frequency, Credit Limit, and more. IsoMap was used to reduce this 19-dimension space to a 2-dimension space. The k-means clustering of this data was optimized using the *Elbow Method* to determine the number of k-means clusters used.



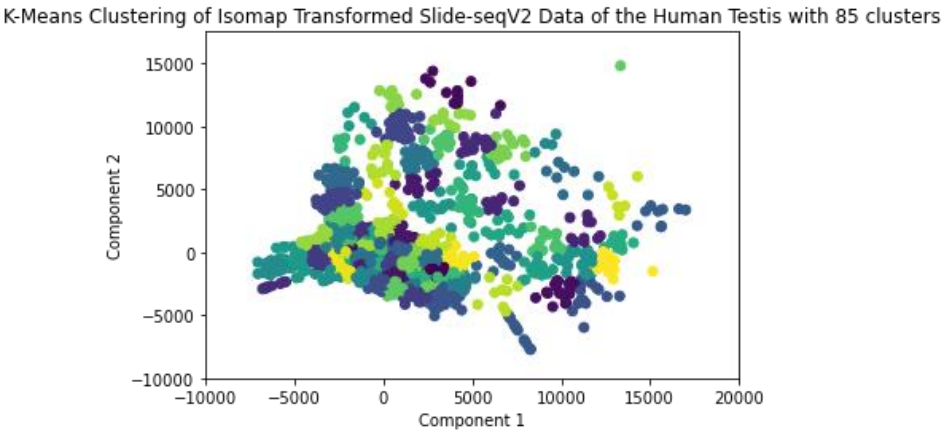
Example 2: Fractional Abundances of Cells Classified within 2.5 Hours of Movement

This dataset presents the fractional abundances of cells within 2.5 hours of *x*-*y* motility tracking under different conditions. There are several cell types tested under different conditions included in this dataset including ovarian cancer cells, germinal center b cells, human dermal fibroblasts, and pediatric lung cells. The numbers of clusters to be used for k-means clustering of this data was also optimized using the *Elbow Method*.



Example 3: Slide-seqV2 Data of the Human Testis

The Slide-seqV2 dataset summarizes the testing of human testis. Slide-seq specifically analyzes and harvests data regarding the spatial organization of different cells in different parts of the tissue. This dataset presents the genome classifications for over 21,000 human genomes and ~6000 targeted genes. IsoMap was used to reduce this to a 2-dimensional dataset and the number of k-means clusters used was determined through the *Elbow Method*.



D. t-Distributed Stochastic Neighbor Embedding

D1. Introduction

T-distributed stochastic Neighbor Embedding (T-SNE) is an unsupervised, nonlinear technique for visualizing high dimensional data onto low-dimensional spaces [6]. T-SNE is different from linear dimension reductionality methods because it preserves small pairwise dimensions and local similarities, while, linear techniques, like PCA, preserve large pairwise distances and tends to maximize variance along certain directions.

D2. Theory

T-SNE was first proposed by Laurens van der Maaten and Geoffrey Hinton in 2008 [6]. Their algorithm is dependent on the Stochastic Neighbor Embedding model which was proposed by Hinton and Roweis in 2002. A discussion of Stochastic Neighbor Embedding will help lay the foundations for T-SNE.

Stochastic Neighbor Embedding

Consider a dataset represented as a matrix $D \in R^{n\,x\,m}$, where *n* is the number of data records in the dataset and *m* is the dimension of each data vector in the dataset.

The first step in Stochastic Neighbor Embedding is to compute the pairwise Euclidean distances between each data vector in *D* [9]. These distances will be used to construct conditional probability distributions which represent similarities between each data vector. For each data vector, $x_i \in D$, a gaussian distribution is generated with mean $\mu = x_i$ and variance σ_i [9]. This conditional probability distribution can be denoted by $P_{j|i|}$. $P_{j|i|}$ gives the probability of a data vector *i* picking another data vector *j* as its neighbor.

Consider the Euclidean distance $\left|\left|X_i - X_j\right|\right|^2$ and the functional form of $P_{j|i|}$ as follows:

$$P_{j|i|} = \frac{\exp\left\{-\frac{\left||x_i - x_j|\right|^2}{2\sigma_i^2}\right\}}{\sum_{k\neq i}\exp\left\{-\frac{\left||x_i - x_k|\right|^2}{2\sigma_i^2}\right\}}$$

The second step is to define $[y_1, \dots, y_n]$ which represents the low dimensional vectors of the data $[x_1, \dots, x_n]$ [6]. A similar conditional probability distribution can also be constructed for $[y_1, \dots, y_n]$. Define $q_{j|i|}$as the probability that data vector y_i will choose $y_j$as its neighbor.

$$q_{j|i|} = \frac{\exp\left\{-\left||y_i - y_j|\right|^2\right\}}{\sum_{k\neq i}\exp\left\{-\left||y_i - y_k|\right|^2\right\}}$$

Since we are only interested in the pairwise probabilities, the algorithm sets $q_{j|i|} = 0$.

The next step is to determine the low dimensional data representation that minimizes the difference between $q_{j|i|}$and $p_{j|i|}$.

A common mapping function which measures how different the probabilities are is the *Kullback – Leibler Divergence* [6]. Stochastic Neighbor Embedding minimizes the sum of *Kullback – Leibler divergences* over all data vectors using gradient descent. The cost function is given by the following:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Since the *Kullback – Leibler divergence* is not symmetric, different types of errors, either distance wise or computational, in the low dimension space are not equally weighted [9]. This can cause some computational errors or runtime issues.

Given that *D* can vary greatly and is often not balanced or uniform, a single best fit σ_i may not exist, but an iterative technique can be applied to find an approximate optimal σ_i [9]. Generally, high density data has a lower variance while sparse data has a greater variance. SNE performs a binary search for σ_i. This binary search produces iterative values of $P_i$with a fixed perplexity. Perplexity is an optimization metric which gives the effective number of neighbors the conditional distribution considers [6]. Perplexity is defined as the following:

$Perp(P_i) = 2^{H(P_i)}$, where $H(P_i) = -\sum p_{j|i} \log_2 p_{j|i}$ (Shannon Entropy)

The minimization of the cost function from above is computed using **gradient descent**. The partial derivative of the function with respect to y is given as follows:

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

The gradient descent is initialized by sampling map points randomly from an isotopic Gaussian with a relatively small variance, centered around the origin. The process is iterative and terminates upon reaching a global minimum [6]. The resulting solution is a set of lower dimensional data points denoted above as $[y_1, \dots, y_n]$.

Disadvantages of Stochastic Neighbor Embedding

One of the major disadvantages of SNE is that the cost function is extremely expensive to minimize using gradient descent. [9] This is true because SNE does not involve any convex optimization problems. The algorithm requires a reasonable amount of initial gaussian noise and optimal learning rates for gradient descent. The combination of noisy data and the initial gaussian fittings require smaller step sizes and thus a smaller learning rate in gradient descent. To combat this issue, optimal parameter values must be used to fit the model. This, however, requires iterations of the optimization algorithm which takes up data storage and run time.

Another disadvantage of SNE is the Crowding Problem [9]. Suppose we have a set of data points that lie on an n-dimensional manifold, where n > 10, and is embedded in a space of higher dimensionality. The pairwise distances of a 2-dimensional mapping cannot accurately measure the distance or "similarity" of the data points on the manifold. Consequently, the distributions of these distances can vary greatly between the higher and lower dimension spaces [9]. Another consequence is that large distances in the high dimension space will be mapped to proportionally smaller distances in the lower dimension space.

Modifications of SNE for T-SNE

To combat the Crowding Problem, Maaten and Hinton proposed two critical modifications to SNE [6]. The first modification involves creating a symmetric version of the cost function in SNE with more efficient gradients.

A symmetric cost function is extracted from the idea that an alternative to minimizing the sum of the *Kullback-Leibler* divergences between the conditional probabilities $p_{j|i}$ and $q_{j|i}$is to minimize a single *Kullback-Leibler divergence* between a joint probability distribution, *P* and a joint probability distribution *Q,* where *P* is the distribution over the high dimension space and *Q* is the distribution over the low dimension space [6]. The cost function is then updated as such:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

This cost function is categorized as symmetric because $p_{ij} = p_{ji}$and $q_{ij} = q_{ji}\ \forall i, j$. The individual conditional probabilities in the high dimension space are defined as follows. It is important to note that these probabilities are once again extracted using the gaussian distribution using the same techniques for determining the variance:

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)}{\sum_{k \neq l} \exp\left(-\|x_k - x_l\|^2 / 2\sigma^2\right)}$$

The second modification is that the joint probabilities in the lower dimension space are determined from the Student t-distribution (with one degree of freedom) instead of the gaussian distribution [6]. The Student t-distribution has much heavier tails compared to the gaussian distribution. This helps map the appropriate distances from the high dimension to the low dimension spaces. The joint probabilities of the lower dimension, applied above, are defined as follows under this new distribution:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

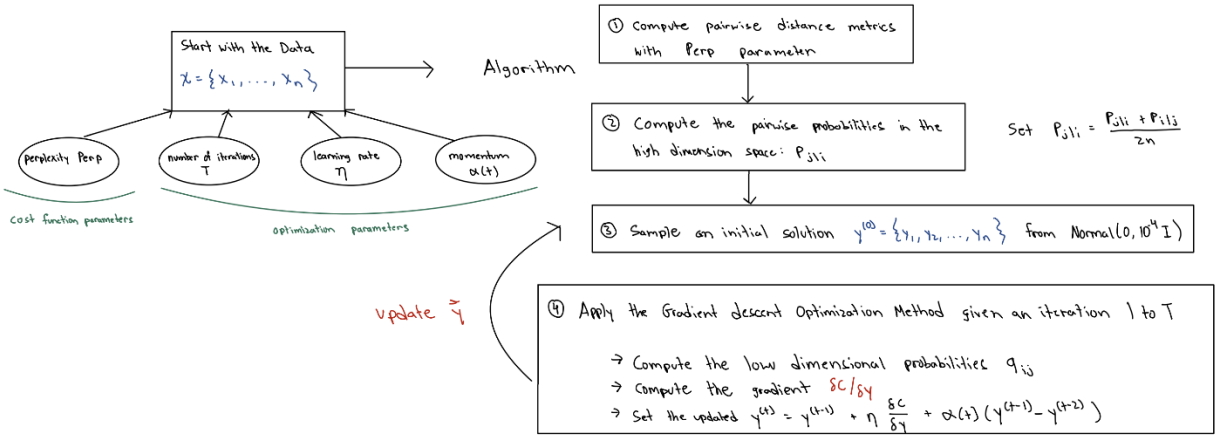
An advantage of using the symmetric SNE terms with the combinations of the gaussian and Student t-distributions is that they produce a much simpler gradient, which is faster to compute. The gradient of the updated cost function is given as follows:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}$$

Gradient descent is once again applied using the new cost function to determine the low dimensional representation of the data.

D3. Algorithm

One of the most common computational methods to implement T-SNE on real-world, large-scale datasets is Maaten's "Simple TSNE implementation" with standard gradient descent.



The first optimization method suggests adding a momentum term to help reduce the number of iterations required by Gradient Descent [6]. In addition, an adaptive learning rate can be implemented. In this scheme, the learning rate for gradient descent increases in the direction the gradient is more stable. Another method, more broadly referred to as "early compression", forces all the mapped points to stay closer together at the beginning of the optimization [9].

This is implemented by adding an additional L2-penalty term to the cost function that is proportional to the sums of squared distances of the mapped points from the origin.

Run Time Analysis

Given that T-SNE has no analytical solution for the underlying optimization problem, iterative techniques must be deployed. This often increases the computational times of the algorithm. One component when analyzing the run time of T-SNE is to consider the pairwise distance computations. This step often increases the time complexity of the algorithm: $O(n^2)$, where n = number of data vectors. Furthermore, given the learning rate and perplexity parameter, the exact run times can vary [6]. However, a widely used modification of t-SNE exists to reduce the time complexity: Barnes Hut t-SNE. This method uses hierarchal trees to estimate the pairwise distances. This results in a time complexity of $O(n \log n)$.

Toolbox Implementations

The *scikit-learn* implementation of t-SNE provides a variety of parameters that allow for fine-tuning of the t-SNE algorithm to produce the best possible embedding.

- n_components: the number of dimensions in the low-dimensional space.
- perplexity: a measure of the balance between preserving local and global structure in the embedding; usually between 5 and 50.
- learning_rate: the step size at each iteration of the gradient descent optimization.
- n_iter: the number of iterations of the gradient descent optimization.
- early_exaggeration: a parameter that controls the degree of exaggeration of attractive forces between nearby points in the early stages of the optimization; usually between 2 and 10.
- metric: the distance metric used to calculate pairwise distances between data points in the high-dimensional space. The default metric is Euclidean distance.
- init: the initialization strategy for the low-dimensional embedding. The default strategy is to use a random initialization.

Run Time, Parameter Analysis

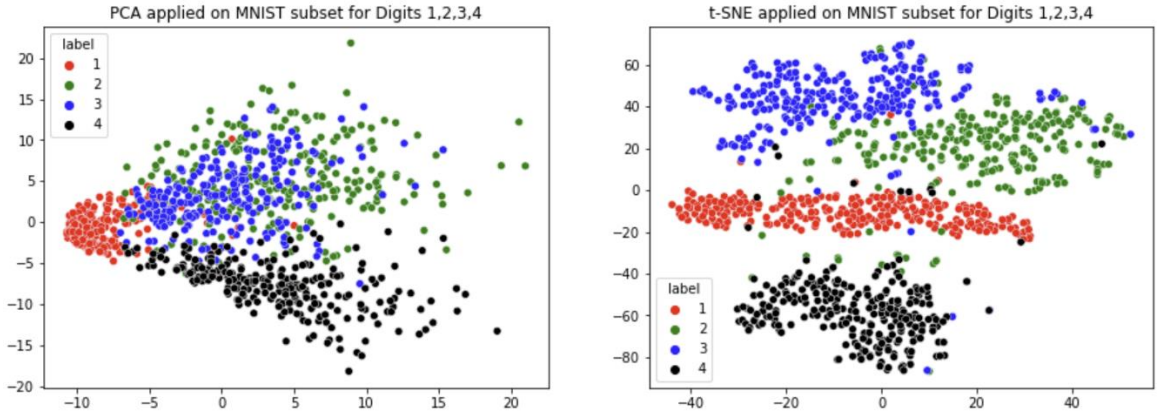
To better understand the time complexity of the algorithm as a function of the parameters, an analysis of some of the critical parameters can be conducted. This can help motivate optimal choices for the parameters to accurately weigh the tradeoff between accuracy and run time.

A subset of the MNIST dataset will be used for this analysis. The MNIST dataset is a collection of 20 by 20-pixel handwritten images used for image classification tasks. This subset contains 4 label digits {1,2,3,4} with approximately 250 datapoints in each cluster. Dimensionality reduction methods are often deployed on this dataset to better visualize the clusters each digit form.

The following set of visualizations highlights the effect each hyperparameter of t-SNE has on the run time of the algorithm.

It can be determined that as perplexity and the number of iterations in gradient descent increase, the run time significantly increases but the accuracy of clustering increases. On the other hand, the value of the learning rate has limited impact on the run time of the algorithm.

Optimal parameters were extracted from this analysis considering the run time – accuracy tradeoff (perplexity = 25, n_iter = 2000, learning_rate = 200). The following visualizations show the two largest PCA components and t-SNE components applied on this data:

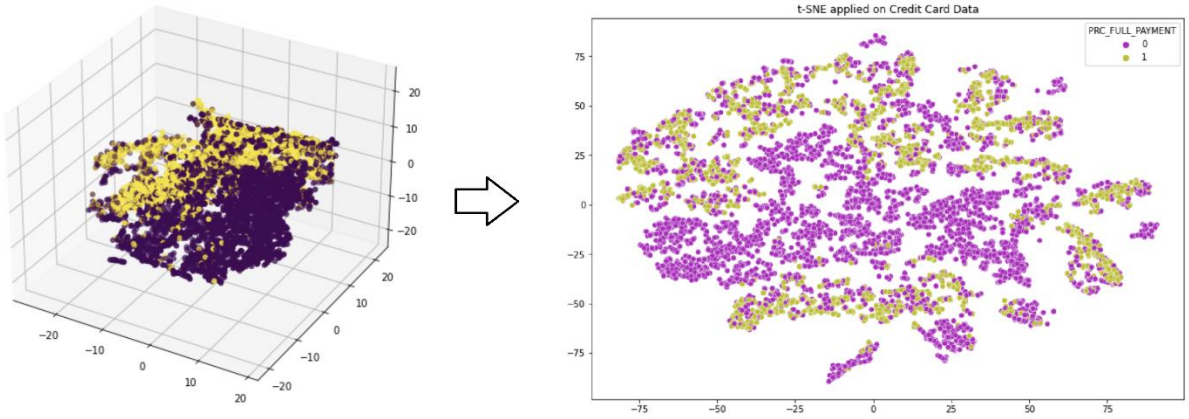


D4. Application

T-SNE is often used to model and analyze large scale financial data. For example, loan and credit data often contain a large number of features. By applying t-SNE to these high-dimensional datasets, it is possible to identify clusters of similar customers or transactions, which can be useful for various purposes such as fraud detection, customer segmentation, and risk management. Additionally, t-SNE can help to visualize the distribution of high-dimensional data in a low-dimensional space, providing insights into the underlying structure of the data. This can reveal non-linear relationships between different features and identify outliers that may indicate potential fraud or other irregularities.

Example 1: Credit Card Data

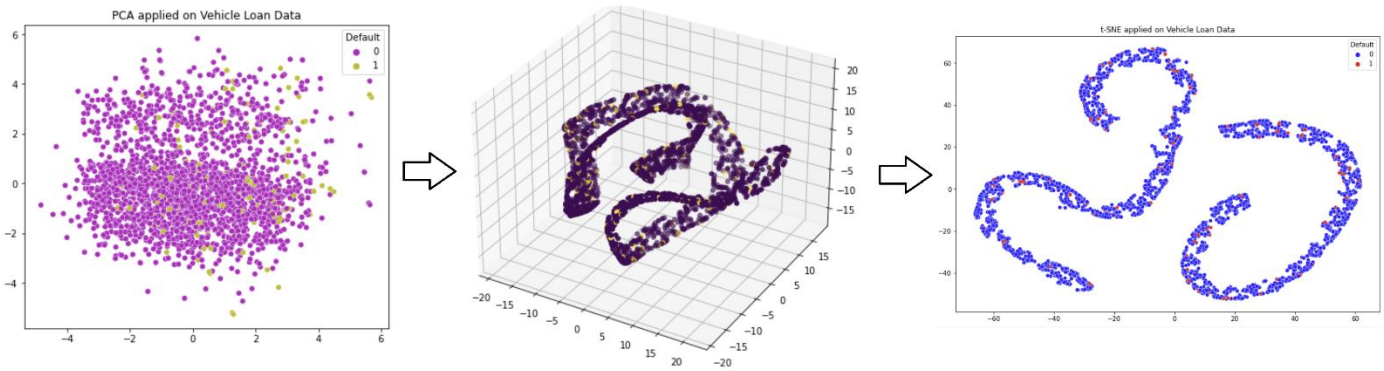
This credit card dataset summarizes the usage behavior of approximately 9000 active credit card holders. There are 18 credit card attributes in the dataset, including Balance, Balance Frequency, and Credit Limit. T-SNE is applied on this dataset to reduce the 18-dimension space to 3-dimensions and 2-dimensions.



Example 2: Loan Default Data

A non-banking financial institution (NBFI) provides financial services like those of banks, such as investment, risk pooling, contractual savings, and market brokering. An NBPC is currently facing profitability issues due to an increase in defaults on vehicle loans. Thirty attributes are contained in this dataset and are used to cluster customers by those who will or will not default on their loans. The dataset contains 121,800 records from customers in a single year.

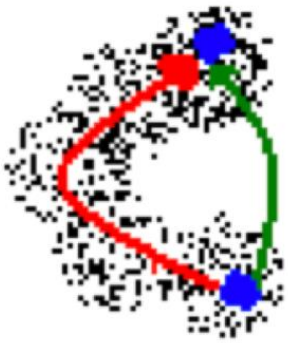
We applied PCA and t-SNE (in 2d and 3d) on this dataset and noted how the attributes when mapped to a lower dimension space don't best quantify the default label. This can suggest to the NBFI that other experiments or attributes should be looked at. The visualization, however, does indicate an underlying natural cluster. It would be worthwhile for the NBFI to investigate the source of this cluster.



E. Diffusion Maps

Low dimensional data transformed to higher dimensions lies on a manifold, a geometric structure that can be non-linear. A diffusion map is a strategy for finding this lower dimension embedding. [13] The Euclidean distance between points resembles diffusion distance in the original feature space. Diffusion matrices have each entry p_{ij} represent the probability of jumping between data point i to j. This is the connectivity between two points, and can be thought of as the chance of jumping from one point to another on a random walk. A diffusion metric is calculated as: $D_t(X_i, X_j)^2 = \sum_{u \in X} |p_t(X_i, u) - p_t(X_j, u)|^2 = \sum_k |P_{ik}^t - P_{kj}^t|^2$

The probabilities P^t are calculated for increasing t as the diffusion process progresses. As t increases, the probability of following a path along the geometry of the data set is more likely as that is where points should be denser and more connected. In the figure below, the red path's probability increases with t because of the short jumps that connect start to end. On the other hand, the green path remains improbable as the value of t rises.

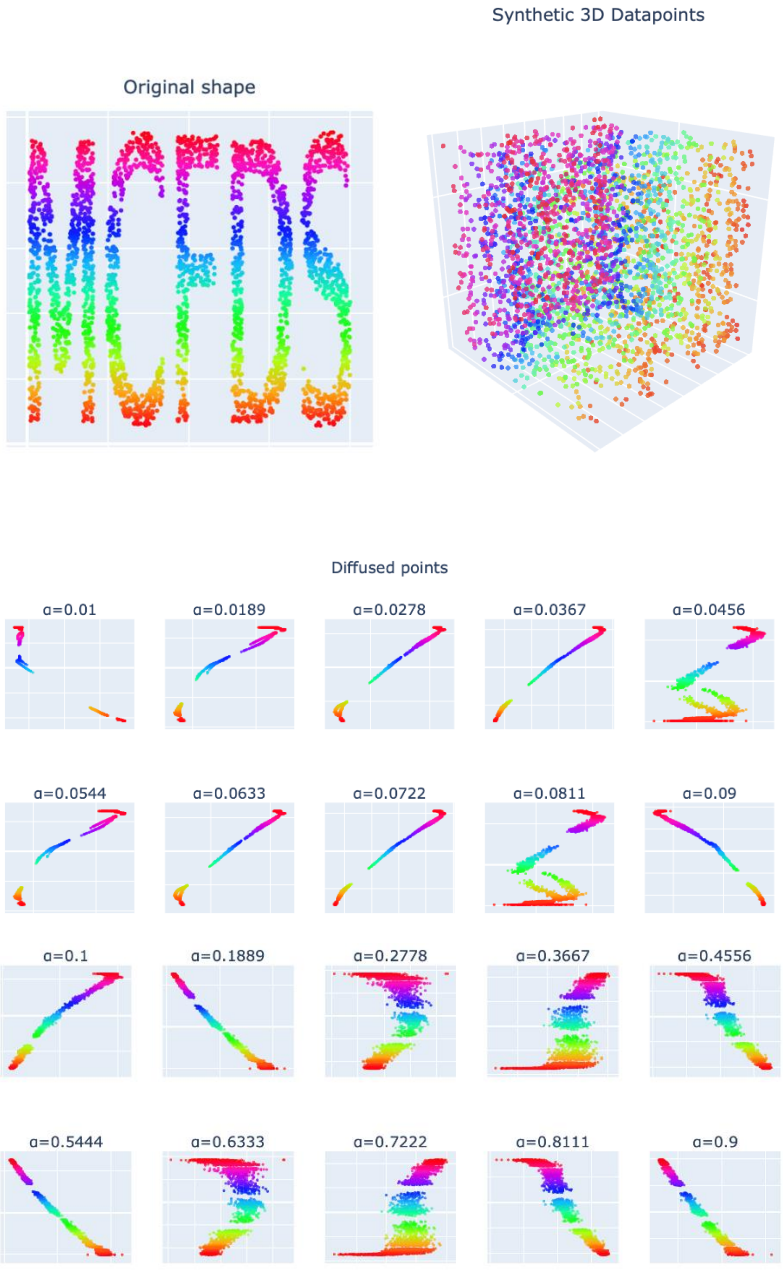


So, as the diffusion process occurs, paths along the underlying geometry will have the highest contribution to the diffusion distance. It takes a lot of computational power to calculate these diffusion distances. Instead, the data can be mapped into Euclidean space according to the diffusion metric. The diffusion distance in this new diffusion space becomes the Euclidean distance. A diffusion map maps coordinates to diffusion space, and so it reorganizes the data based on the diffusion metric. This map is used to reduce dimensions, as afterwards fewer coordinates are needed to describe the same data points. It preserves intrinsic geometry as well.

One application of diffusion maps is in single cell genomics. [4] During developmental processes, cells might not be clustered distinctly but it would still be of interest to profile and classify them. In this situation, the many cells make up a high dimensional data set. Analyzing random walks between cells near each other with a Gaussian kernel can reveal the structure of the underlying process. In essence, the local diffusion of each cell is measured. The Markovian transition matrix for the cell holds the probabilities of transitioning to any other cell and is proportional to a Gaussian kernel. All of the walks are then superimposed to create a diffusion

map, showing the high probability paths on a manifold. The manifold can also be projected onto the first eigenvectors of the Markov transition matrix, called the diffusion components. This is not a linear projection, unlike covariance matrices from PCA.

To illustrate an example of diffusion mapping, a python toolkit was used in a top-down approach. [14] It first takes a 2D image and assigns a random value for its third dimension, thus generating a 3D distribution of points from it. Then, the distribution is diffused down to 2D, to test if it can capture the original structure. Alpha is a parameter that scales the exponent used for left normalization when creating the diffusion map. Alpha values of 0.01-0.9 were tested as shown below:



F. Relation to the Course

Manifold learning combines many concepts taught in class, such as Laplacian eigenmaps, spectral clustering, and k-means. Some of the key concepts and their applications are highlighted below:

Norms: Many manifold learning methods involve critical computations of the norms of data vectors. For example, t-SNE considers the standard 2-norms for computing distance.

PCA: Principal component analysis is also a dimensionality reduction technique. As discussed, manifold learning is effective for non-linear data, whereas PCA is a linear technique.

Markov processes: The steady-state probability of the Markov Chain is used to calculate the distances in the lower dimension embedding during diffusion mapping. The diffusion matrix is a Markov matrix.

Eigenvectors/eigenvalues: Laplacian eigenmaps use the first eigenvectors to describe the basis of reduced dimensions. Diffusion mapping uses the first eigenvectors as diffusion components, the dominant aspects of the transition matrix.

G. Conclusion

Based on the information presented, it is clear that manifold learning algorithms are a vital tool for understanding complex data with intricate, non-linear structures. The three primary algorithms covered in this report, t-SNE, Laplacian Eigenmaps, and Isomaps, each possess their unique benefits and limitations that make them suitable for a wide range of applications. Additionally, diffusion maps are another powerful manifold learning algorithm that can identify and extract valuable features from high dimensional data. This is especially applicable with genomic data and single cell analysis.

Despite manifold learning's potential to uncover hidden patterns and structures in data, it is important to note that it is not without its limitations. Factors such as overfitting and hyperparameter sensitivity can negatively impact results, and appropriate measures must be taken to mitigate these risks given the situation at hand.

H. References

[1] Balasubramanian, Mukund, and Eric L. Schwartz. "The Isomap Algorithm and Topological Stability." *Science*, vol. 295, no. 5552, 2002, pp. 7-7.

[2] Barahimi, Farshad, et al. "Multi-point dimensionality reduction to improve projection layout reliability." *ArXiv*, 15 Jan. 2021, www.semanticscholar.org/paper/Multi-point-dimensionality-reduction-to-improve-Barahimi-Paulovich/28de820e0863e1de4dfa33e52d9e226db0256935. Accessed 9 Apr. 2023.

[3] Belkin, Mikhail, and Partha Niyogi. "Geometric Methods and Manifold Learning." *Ohio State*, web.cse.ohio-state.edu/mlss09/mlss09_talks/3.june-WEN/mlss09.pdf. Accessed 9 Apr. 2023.

[4] Broad Institute. "MIA: Fabian Theis, Reconstructing trajectories and branching lineages in single cell genomics." 26 June 2017, *YouTube*, www.youtube.com/watch?v=gVAEk2zBf_E. Accessed 9 Apr. 2023.

[5] Horaud, Radu. "A Short Tutorial on Graph Laplacians, Laplacian Embedding, and Spectral Clustering." *California State University*, csustan.csustan.edu/~tom/Clustering/GraphLaplacian-tutorial.pdf. Accessed 9 Apr. 2023.

[6] Huroyan, Vahan, et al. "Embedding Neighborhoods Simultaneously t-SNE (ENS-t-SNE)." *ArXiv*, 24 May 2022, www.semanticscholar.org/paper/Embedding-Neighborhoods-Simultaneously-t-SNE-Huroyan-Navarrete/3aacde218e002f8fdfc49572fd9e41f7da426dcc#citing-papers. Accessed 9 Apr. 2023.

[7] Kovan, Ibrahim. "Multidimensional Scaling (MDS) for Dimensionality Reduction and Data Visualization." *Medium*, 15 Oct. 2021, towardsdatascience.com/multidimensional-scaling-mds-for-dimensionality-reduction-and-data-visualization-d5252c8bc4c0. Accessed 9 Apr. 2023.

[8] Li, Bo, et al. "A survey on Laplacian eigenmaps based manifold learning methods." *Neurocomputing*, vol. 335, 2019, pp. 336-351.

[9] Maaten, Laurens V., and Geoffrey Hinton. "Visualizing Data using t-SNE." *Journal of Machine Learning Research*, vol. 9, no. 86, 2008, pp. 2579-2605, www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf. Accessed 9 Apr. 2023.

[10] "Mathematics | Graph Theory Basics - Set 2 - GeeksforGeeks." *GeeksforGeeks*, 3 Dec. 2021, www.geeksforgeeks.org/mathematics-graph-theory-basics/. Accessed 9 Apr. 2023.

[11] Pascual, Angela. "Advanced methods for dimensionality reduction and clustering: Laplacian Eigenmaps and Spectral Clustering." 2010, www.semanticscholar.org/paper/Advanced-methods-for-dimensionality-reduction-and-Pascual/1777ca4bb9fe1cd0f85e24857488339eb2e6a7dc#related-papers. Accessed 9 Apr. 2023.

[12] Pedregosa, Fabian, et al. "2.2. Manifold Learning." *Scikit-learn*, 2011, scikit-learn.org/stable/modules/manifold.html. Accessed 9 Apr. 2023.

[13] Porte, J., et al. "An Introduction to Diffusion Maps." *Colorado School of Mines*, Nov. 2008, inside.mines.edu/~whereman/talks/delaPorte-Herbst-Hereman-vanderWalt-DiffusionMaps-PRASA2008.pdf. Accessed 9 Apr. 2023.
[14] Rahulrajpl. "Diffusion Map for Manifold Learning." *Kaggle*, 14 Mar. 2020, www.kaggle.com/code/rahulrajpl/diffusion-map-for-manifold-learning. Accessed 9 Apr. 2023.
[15] Ribeiro, Bernardete, et al. "Supervised Isomap with Dissimilarity Measures in Embedding Learning." *Lecture Notes in Computer Science*, 2008, pp. 389-396.
[16] Tenenbaum, Joshua B., et al. "A Global Geometric Framework for Nonlinear Dimensionality Reduction." *Science*, vol. 290, no. 5500, 2000, pp. 2319-2323.
[17] Yang, Huan, and Haiming Li. "Implementation of Manifold Learning Algorithm Isometric Mapping." *Journal of Computer and Communications*, vol. 07, no. 12, 2019, pp. 11-19.
[18] Ye, Andre. "Manifold Learning [t-SNE, LLE, Isomap, +] Made Easy." *Medium*, 14 Aug. 2020, towardsdatascience.com/manifold-learning-t-sne-lle-isomap-made-easy-42cfd61f5183. Accessed 9 Apr. 2023.