Diffusion Maps

Patrick H. Rupp

E-mail: prupp1@jhu.edu

Abstract

Diffusion maps are a technique that is increasing in popularity for meeting the curse of dimensionality problem. Data sets are growing both vertically and horizontally as problems are becoming more complex. This paper attempts to demonstrate the power that Diffusion Maps have at reducing the dimensions required to represents the shape of the original data by identifying the underlying manifold. This uses the diffusion maps on four different data sets including a Helix, Sine Wave, Torus, and Spiral in 4D and reduces the data to 2D. The pydiffmap did an exceptional job at identifying the underlying manifold with a fast and memory efficient algorithm versus the KD Nuggets implementation which is not scalable for both computation time and memory usage reasons. This test was conducted on a Ubuntu 18.04 virtual machine using Oracle Virtual Box and the shell scripts can only run on linux images.
Code: https://github.com/PHRupp/sde

1. Background

Data is growing at an alarming rate as well as becoming more complex in nature. It is becoming common to have datasets with high number of dimensions. Analyzing this type of data is difficult and cumbersome often making it impossible to find the underlying correlations in the data.

Dimensionality reduction is a technique that attempts to identify dimensions which are correlated and reduce them to a smaller number of dimensions that can still represent the original shape of the data. There exists many different techniques each with their own advantages and limitations including both linear and non-linear approaches.

1.1 PCA

One of the most common dimensionality reduction technique is Principle Componenet Analysis (PCA). The PCA technique uses an approach that identifies features that are linearly correlated and attempts to reduce those dimensions by projecting the data in lower dimenions over new features axes. PCA is widely used within exploratory data anlaysis because visualizing correlations in high dimensional data is a difficult problem.

1.2 Diffusion Maps

Another technique that is becoming popular is called Diffusion Maps. This technique attempts to perform dimensionality reduction by attempting to find an underlying non-linear relationship in the data and projects the data to the lower dimension.

The special highlight with the diffusion map technique is that it is a non-linear approach unlike the PCA which is a linear approach. In practice, it is hard to find linearly correlated data which makes the diffusion maps a good option. It can perform the non-linear approach by assuming there is some type of lower dimension manifold tying the data together.

2. Dimension Reduction Experiment

The first experiement will be to test multiple differnet data sets to see how well the different dimensionality reduction techniques perform. The two main experiements will consist of using various implementation of diffusion maps to reduce the data to lower dimensions and the compare the differnet implementations of the diffusion maps algorithms to see how they perform in both time and space complexity.

The first experiment consists of reducing four differnet data sets. The data sets will be repsented in differnet mathematical shapes such as the helix, sine wave, torus, and spiral. These shapes were generated with randomness in all dimensions. In addition, the plots have exactly 1000 points across all four tests.

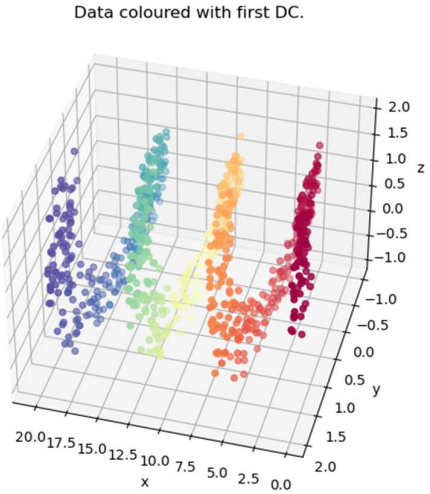

Figure I – Displays the original 4D data set in the form of a helix with noise. The helix is visible visually in 3D and the 4th dimenion is linear with the 'z' axis. The underlying manifold in this case would be the helix shape.

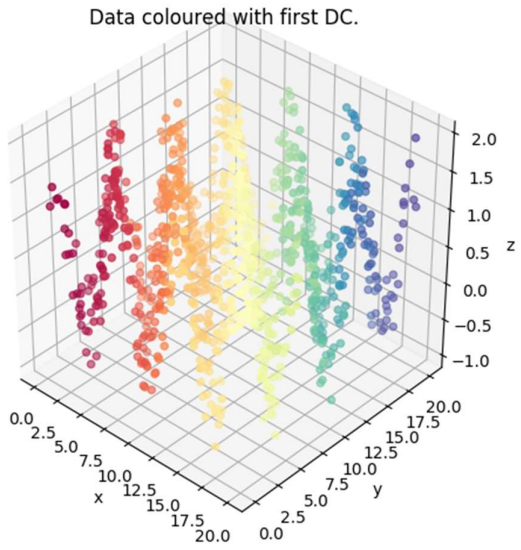

Figure II – Displays the original 4D data set in the form of a sine wave with noise. The sinewave is visible visually in 3D and the 4th dimenion is a "termperature" value corresponding to cos(x). The underlying manifold in this case would be the sine wave.

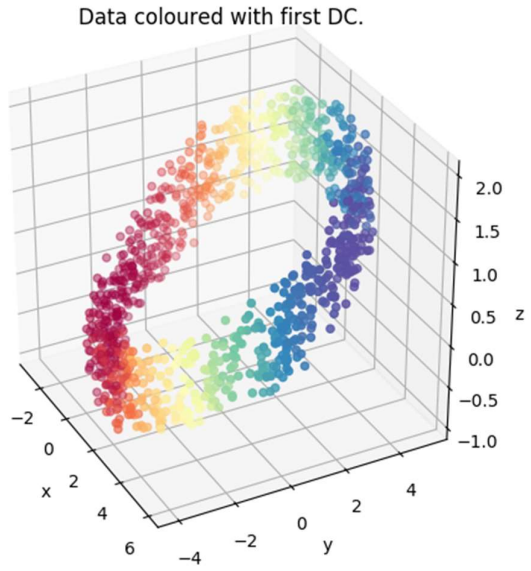

Figure III – Displays the original 4D data set in the form of a tilted torus with noise. The torus is visible visually in 3D and the 4th dimenion is a "termperature" value corresponding to relationship with y. The underlying manifold in this case would be the torus.

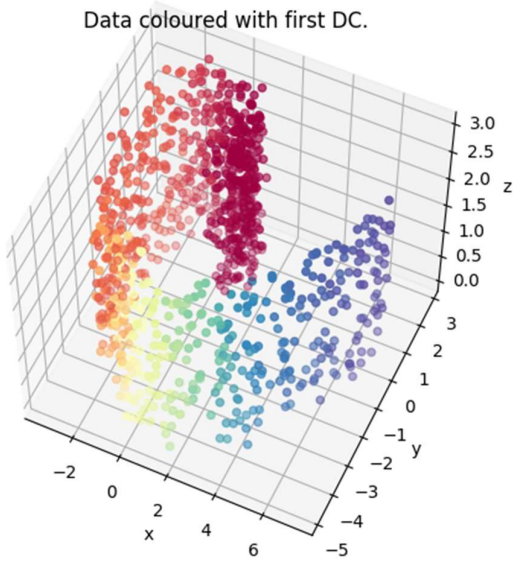

Figure IV – Displays the original 4D data set in the form of a spiral with noise. The spiral is visible visually in 3D and the 4th dimenion is a "termperature" value corresponding to cos(x). The underlying manifold in this case would be the spiral. The spiral is also commonly known as the "swiss role".

2.1 Diffusion Maps

For our diffusion maps experiment, two different implementation are used. One implementation used can be found within the pydiffmap python library whereas the other

was developed on KDNuggets website for an article detailing diffusion maps [5].

For this experiment, the parmeters held constant were the following:

- number of dimensions = 2
- alpha = 3.0
- distance metric = 'euclidean'
- random weight = 1.0
- neighbors = 10

2.1.1 pydiffmap. This library was developed to provide users with the ability to build diffusion maps quickly and easily in addition includes built in plotting mechanisms. This library utilizes the sklearn library for a lot of the processing under the hood. As of 8/23/2020, this was the only reliable diffusion map library that was publicly released for python community that the author could find.

The pydiffmap library provides the user with the ability to manipulate various parameters and settings which would affect the performance of the diffusion map. These parameters and settings includes things such as number of dimensions, distance metric (eg. 'euclidean', 'mahalanobis'), the alpha parameter, number of neighbors, and more.

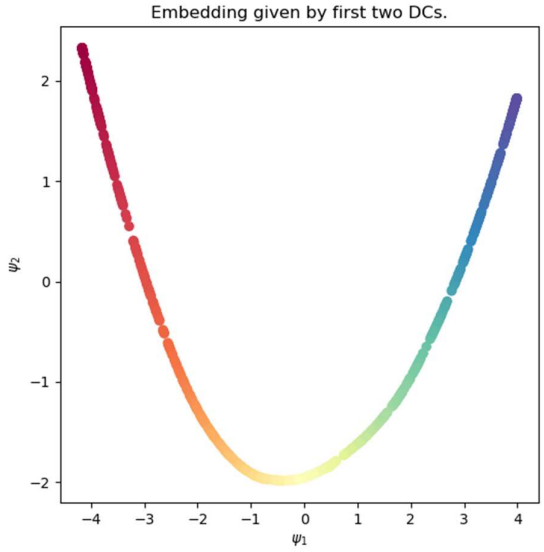

Figure V – Displays the reduced data from the pydiffmap implementation of the diffusion maps of the **helix**. This technique found a lower dimenion manifold reminescent of a quadratic function. The change of color can be correlated with the increase in value of the x-axis verifying that this data is indeed compresed to 2 dimensions from 4.

The pydiffmap was able to turn the 4D helix into 2D plot. Since the manifold can be thought of a string in the form of a helix, then we would expect to see the reduced data in the form of a string of some form. In the above, we see that the shape that the string takes on is in the form of a quadratic funciton. It is underdetermined entirely why the shape is a quadratic, but it is encouraging to see that the algorithm picked up on the underlying manifold.

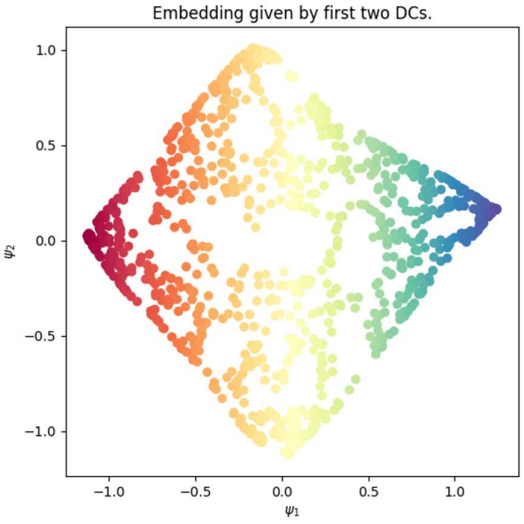

Figure VI – Displays the reduced data from the pydiffmap implementation of the diffusion maps of the **sine wave**. This technique was able to flatten the sine wave from the 3D ripples to the 2D sheet that the data represented.

The pydiffmap was able to turn the 4D sine wave into 2D plot. The manifold can be thought of a sheet in 2D with ripples in the 3rd and 4th dimension. The diffusion map was able to identify the underlying manifold (aka. The "sheet") from the Figure VI. As one can see from Figure II, the ripples in the manifold are quite large, and yet the diffusion map was able to account for this well.

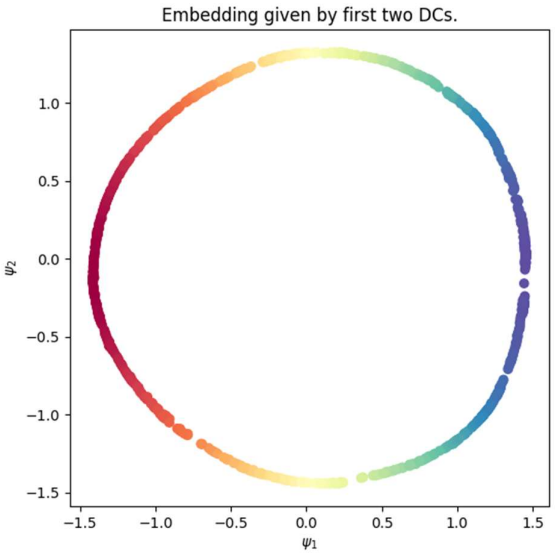

Figure VII – Displays the reduced data from the pydiffmap implementation of the diffusion maps of the **torus**. This technique found a lower dimenion manifold reminescent of a quadratic function.

The pydiffmap was able to turn the 4D torus into 2D plot. The manifold can be thought of as a donut with the center missing. Figure III clearly shows that the torus is being reduced to a tight 2D circle. In addition, we can see reasonable amount of noise included in the data points and the diffusion map reduced the noise down the underlying manifold quite well. The reduced data does have a slightly more oblong shape than a perfect circle potentially due to the randomness in the data.

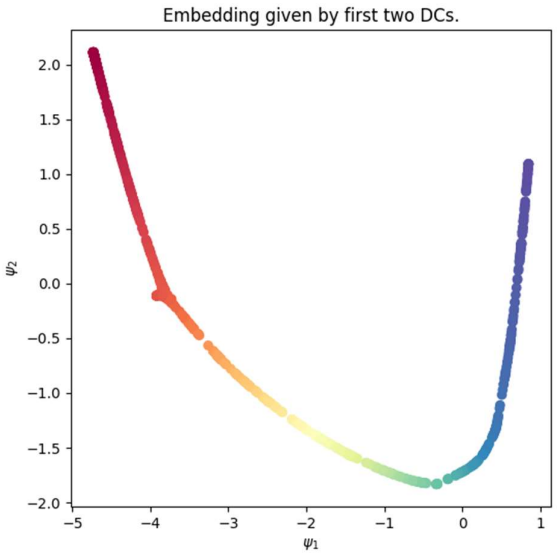

Figure VIII – Displays the reduced data from the pydiffmap implementation of the diffusion maps of the **spiral**. This technique found a lower dimenion manifold loosely reminescent of a quadratic function.

The pydiffmap was able to turn the 4D spiral into 2D plot. The manifold can be thought of as a swiss roll where the points get further away from the origin as we continue to rotate around the origin (see Figure IV). Figure VIII shows that the diffusion map was able to identify some features of the manifold such as the rotation, but it is not entirely clear if the mapping was successful.

2.1.2 KD Nuggets. The article found in resource [5], also gave an example implementation on creating and using diffusion maps. The implementation is theoretically correct, but offers much less capability for changing parameters like the pydiffmap implementation. In addition, there are some implementation concerns that warrant further analysis as detailed in section 3.

The KD Nuggets implementation utilizes all records to build it's map. Meaning, that instead of using a set number of neighbours, the algorithm uses all other points as the neighbours. This creates the cost in both space and complexity which greatly differentiates pydiffmap's implementation from their own. In addition, the results will be different because each point is being influenced by all other points instead of a set number of neighbours. I theorize that we will see maps that are much less clear and less similar then their original raw format.

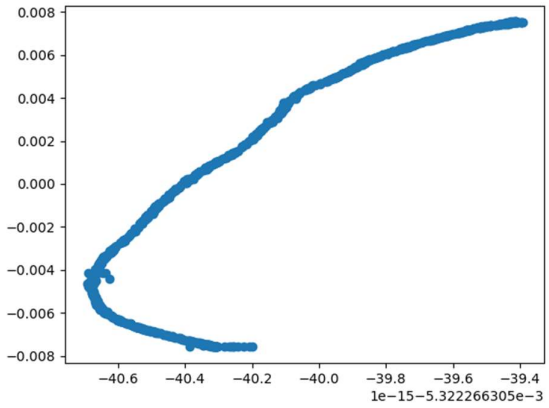

Figure IX – Displays the reduced data from the kdnuggets implementation of the diffusion maps for the **helix** data. This technique found a lower dimenion manifold reminescent of a quadratic function also, but it is less clean.

The KD Nuggets library was able to turn the 4D helix into 2D plot. Since the manifold can be thought of a string in the form of a helix, then we would expect to see the reduced data in the form of a string of some form. In the above, we see that the shape that the string takes on is in the form of a quadratic funciton. This format is much less clear than what we saw with the pydiffmap implementation. It looks like the randomness is more impactful here since the reduced manifold is not as smooth.

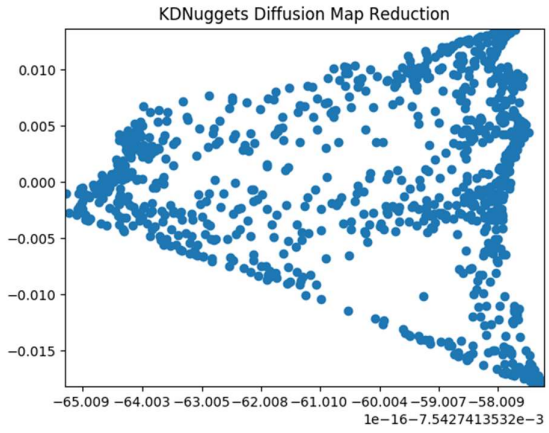

Figure X – Displays the reduced data from the kdnuggets implementation of the diffusion maps for the **sine wave** data. This technique was able to flatten the sine wave from the 3D ripples to the 2D sheet that the data represented, but the shape is not as clean compared to the pydiffmap implementation or the original data set.

The KD Nuggets library was able to turn the 4D sine wave into 2D plot. The manifold can be thought of a sheet in 2D with ripples in the 3rd and 4th dimension. The diffusion map was able to identify the underlying manifold (aka. The "sheet") from the Figure VI. As one can see from Figure X, the diffusion map was able to find the underlying manifold, but it does not represent the rectangular sheet as well as the pydiffmap. In addition, the spacing of points are more drastic in the reduced for here than in pydiffmap's reduced form.

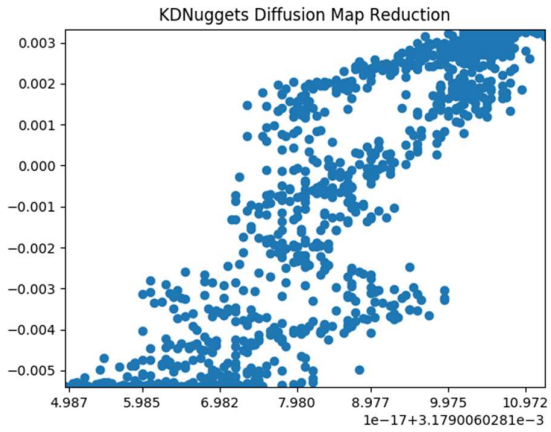

Figure XI – Displays the reduced data from the kdnuggets implementation of the diffusion maps for the **torus** data. This technique was able to flatten the the torus but the output does not resemble the higher dimensional data at all. This might be due to the complex shape influenced by all the points used.

The KD Nuggets library was able to turn the 4D torus into 2D plot. The manifold can be thought of as a donut with the center missing. Figure XI clearly shows that the diffusion map could not identify the underlying manifold. Unfortunately, we declare the KD Nuggets implementation for the torus not succesful. This could potentially be due to the influence of all the points on each other unlike how the pydiffmap only has a set number of neighbors who influence each point.

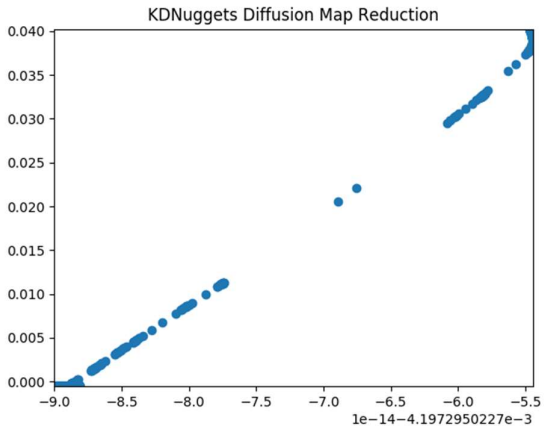

Figure XII – Displays the reduced data from the kdnuggets implementation of the diffusion maps for the **spiral** data. This technique was able to flatten the spiral into a linear plot excluding the ends where there is a high concentration of points. We expect the high concentration one end but not both.

The pydiffmap was able to turn the 4D spiral into 2D plot. The manifold can be thought of as a swiss roll where the points get further away from the origin as we continue to rotate around the origin (see Figure IV). Figure XII shows that the diffusion map was able not able to identify the underlying manifold. The reduced data shows a linear relationship between the new dimensions with gaps in the middle and concentraction of points on both ends. We might expect to see a concentration on one end when the points are close to the origin in the original data.

3. Resource Usage Experiment

While accuracy for a numerical model is of the upmost importance, it is not the only thing that must be considered when evaluating potential solutions. Two main factors that are considered when employing data solutions are Computational comlexity and Space complexity.

Computational complexity deals with the number of computations that the algorithm needs to perform for a given size of data. There exists a relationship between the number of data points given to the model and the time it takes to complete the training of the model.

Space complexity deals with the physical memory space needed by the algorithm to perform the computation. If an algorithm requires more space than the hardware can provide, the algorithm cannot perform its operations on the entire set of data. Hence reviewing both of these points are crucial for identifying the capabilities of various algorithms and methods.

3.1 Computational Complexity

Computational complexity is the number of operations that an algorithm has to use to come to its solution. For instance, the algorithm for a sum of all numbers requires performing

O(n-1) additions. Computational complexity is important to identify for an algorithm because the number of operations has both an impact on the processing time for the solution as well as identifying numerical error due to limited memory storage for floating point values.

Since the pydiffmap's sklearn implementation uses only a set number of neighbours, the data can be stored in a sparse matrix. The matrix multiplication algorithms on a sparse matrix are far more efficient than their full matrix counterparts. This allows pydiffmap's implementation to be both fast and have cheap memory cost.

KD Nuggets implementation utilizes the full data set as the neighbours for all other points. This creates a N by N distance map. Typical matrix multiplication has $O(n^3)$ complexity meaning that as the size of the matrix grows, the time it takes to find the solution is to the power of 3. This is not efficient because when N becomes very large, the time to find the solution grows extremely quickly.

The author attempted to make the KD Nuggets implementation more efficient while maintaining the same underlying methodology. By storing the diagonal matrices as 1D arrays, and performing element-wise matrix multiplication, we were able to avoid all the additional multiplications of elements times 0 due to the diagonal matrix structure.

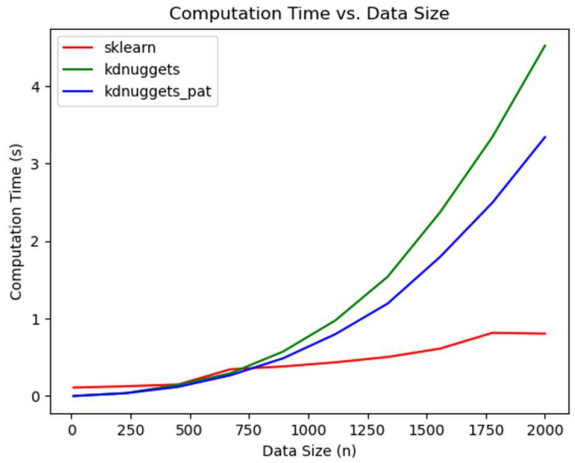

Figure XIII – Displays the computation time as a function of the size of the data for training the diffusion map. The knuggets has the fastest increase over time whereas the sklearn avoids the exponential growth due to not building the distance map between all points.

Figure XIII displays the computation time required the by the algorithms for finding the solution as the data size grew. It is obvious that the original KD nuggets implementation had a very fast growing time complexity. It took over 4 seconds to process only 2000 records. This means that the implementation would not be able to scale up to data sets with more than 10,000 records which is still considered tiny.

The author's modifications reduced the time complexity slightly, but could not fully escape the fast growth. This means that while it will be faster than the original implementation, it too will not be scalable.

The pydiffmap's sklearn implementation shows a strangely unclear shape in the computation time profile. We see at lower data sizes, the algorithm is slower than KD nuggets. However, the algorithm appears to grow at a very slow rate. The algorithm finds a solution in less than 1 second at 2,000 records. In addition, the profile does not indicate to us a very fast growth rate which has the author optimistic about its ability to scale.

3.2 Space Complexity

In addition to time complexity problems, algorithms also suffer from requirements due to memory and storage. Many algorithms require temporary storage for new data or processing in order to reduce the number operations taken. Unfortunately, space complexity has restrictions due to the hardware that the code is being run on.

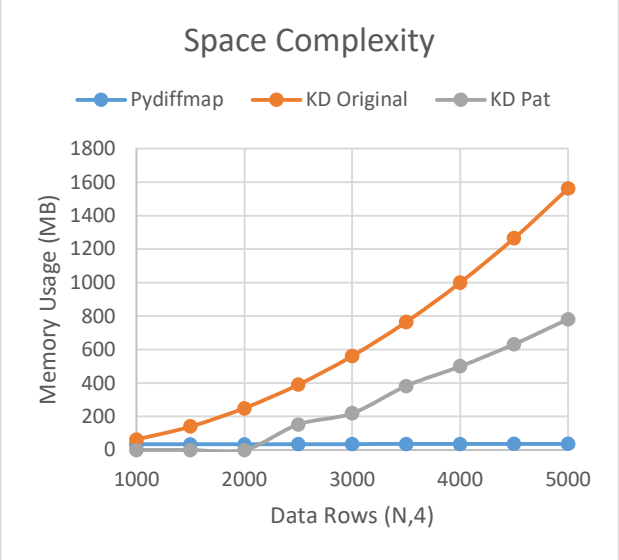

Figure XIV – Displays the memory usage for the differnet algorithms as the number of records within the data increases. This gives us the insight into what the space complexity of the algorithms require. The KD Nuggets is the worst because their algorithm requires storing six N by N matrices whereas the Pydiffmap stores a nearly constant amount due to using a limited set of neighbors in an efficient sparse matrix.

The original KD Nuggets implementation had nearly 6 full matrices in the implementation. Of which, three are diagonal matrices which contain mostly zeros. Diagonal matrix has n^2 elements, but only n are populated whereas the rest of the (n^2-n) are zero. This usage of space to represent so many zeros in a known structure implies a large innefficiency in the implementation.

The author's modifications turned the diagonal matrices into 1D arrays such that only *n* elements are stored in memory. In addition, the algorithm was modified such that not all matrices were stored at the same time if they were not needed anymore. By enforcing the algorithm to unlock the memory when it wasn't needed allowed the space complexity to be reduced. Python inherintely has a garbage collector that identifies when objects are no longer needed and frees up the respective memory. However, the garbage collector runs at a non-trivial time when freeing up memory, making it more efficient for the user to explicitely release the memory on an object.

An odd thing to notice in Figure XIV, the first three tests run, the linux command "pmap" which is used to determine memory consumption by the python process running the code, would not identify the changes in memory usage during the author's implementation. This does not mean that the algorithm used zero memory. Instead, the reader should understand that at the time the measurements from *pmap* were taken, the memory was more likely already freed up. We see that as the data continues to grow that the memory still increases quickly, but less drastically than the original implementation.

Pydiffmap's sklearn implementation barely grows with space complexity. It is the author's understanding that since the data is stored in a sparse matrix, the only real impact to memory usage is based on how big the distance map is which is directly influenced by the number of neighbours within the calculation. Since, this experiment only utilized 10 neighbours the map is still small. However, if the number of neighbours were to increase, then the space complexity would start to grow much more quickly.

4. Conclusion

Diffusions maps have proven to be capability of reducing the dimensionality of a data set by identifying an underlying manifold that the data resides, and representing that manifold in lower dimensional space. In the tests containing a Helix, Sine Wave, Torus, and Spiral, the pydiffmap library using sklearn able to effectively reduce the first three data sets well, and had some questionable results on the Spiral. The KD Nuggets implementation was also able to identify the underlying manifold in the first two cases, but it could not identify the manifold in the second two cases.

The pydiffmap library provided the user with a fast and memory efficient algorithm that has the ability to scale to large data sets. Unfortunately, the KD Nuggets implemenations along with Author's speed and memory improvements could not make amends for the poor usage of computation and space. Thus, the pydiffmap is superior python algorithm for implementing diffusion maps for dimensionality reduction.

References

[1] Banisch, Ralf, Zofia Trstanova, Andreas Bittracher, Stefan Klus, and Peter Koltai. "Diffusion maps tailored to arbitrary non-degenerate Ito processes." arXiv preprint arXiv:1710.03484 (2017).

[2] Porte, Herbst, Hereman, Walt, *An introduction to Diffusion Maps* https://inside.mines.edu/~whereman/talks/delaPorte-Herbst-Hereman-vanderWalt-DiffusionMaps-PRASA2008.pdf

[3] Coifman, Lafon, Lee, Maggionni, Nadler, Warner, Zucker. "*Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps*" https://www.pnas.org/content/pnas/102/21/7426.full.pdf

[4] Wolf, Guy. "Introduction to Data Mining: Diffusion Maps". (2016). http://cpsc445.guywolf.org/Slides/CPSC445%20-%20Topic%2010%20-%20Diffusion%20Maps.pdf

[5] Raj, R., 2020. *Diffusion Map For Manifold Learning, Theory And Implementation - Kdnuggets*. [online] KDnuggets. Available at: https://www.kdnuggets.com/2020/03/diffusion-map-manifold-learning-theory-implementation.html [Accessed 7 July 2020].

[6] Raj, R., 2020. *Diffusion Map For Manifold Learning, Theory And Implementation - Kdnuggets*. [online] KDnuggets. Available at: https://readthedocs.org/projects/pydiffmap/downloads/pdf/master/ [Accessed 11 Aug 2020].