Report for exercise 6 from group H

Tasks addressed:	5
Authors:	Hammad Basit () Antonia Gobillard () Nayeon Ahn () Muhammed Yusuf Mermer () Milena Schwarz ()
Last compiled:	2024–05–18
Source code:	`https://github.com/Hammad-7/MLCMS_Exercises/tree/main/Project`

The work on tasks was divided in the following way:

Hammad Basit () **Project lead**	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Antonia Gobillard ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Nayeon Ahn ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Muhammed Yusuf Mermer ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%
Milena Schwarz ()	Task 1	20%
	Task 2	20%
	Task 3	20%
	Task 4	20%
	Task 5	20%

In our final exercise, we chose to learn and visualize representations for large data sets. Specifically, we want to refer to the paper [McQueen et al., 2016] [7]. We will use different libraries to test their performance against each other and compare it to the package `megaman` that is used in the paper for scalable manifold learning.

Introduction and Motivation

The primary objective of **dimensionality reduction** is to transform high-dimensional datasets, originally characterized by numerous variables, into a concise representation using a minimal set of parameters. This tool plays a crucial role across various disciplines, including information theory (linked to compression and coding), statistics (involving latent variables), as well as machine learning and sampling theory.

Formally, let's assume we have a data matrix $X \in \mathbb{R}^{N\times n}$ with N data points in n-dimensional space. The goal is then to obtain a new representation of the data, as another coordinate matrix $U \in \mathbb{R}^{N\times p}$, ideally with $p << n$. (One can note that for visualization p = 2,3 is necessary.)

Manifold learning techniques aim specifically to capture the intrinsic, lower-dimensional representations of the data, enabling more effective visualization, analysis, and modeling. The quest for meaningful structures in datasets involves extracting relevant features to enhance insight and understanding of the underlying phenomena. The new representation then aims to faithfully describe the data by preserving key quantities, such as local mutual distances. If these methods are particularly useful, it is partly because oftentimes, it is reasonable to assume that high dimensional real-life data lies on a lower-dimensional **manifold** [1].

Diffusion Maps was first introduced by R.R. Coifman and S. Lafon [1] as a non-linear dimensionality reduction algorithm. Since then, the tool has gained a lot of popularity over the years. Often called "Laplacian eigenmaps", it's used to identify significant variables that live in a lower dimensional space while preserving the local proximity between data points. Even though dimensionality reduction algorithms include PCA (Principle Component Analysis), Isomap, LLE (Locally Linear Embedding), and t-SNE (t-Distributed Stochastic Neighbor Embedding), for the purpose of our review, we will focus our interest in this paper on Diffusion Maps.

Having covered some essential aspects of dimensionality reduction, let's return to the mathematical formalism introduced earlier to clarify the Diffusion Maps method. Familiarity with the related concepts and algorithmic steps will be beneficial for the upcoming discussions.

Given a dataset X with N data points $x_1, x_2, \ldots, x_N$ in a high-dimensional space $\mathbb{R}^n$, we want to know how similar two data points are. This can be measured by the **kernel function** $k : X \times X \to \mathbb{R}$, which has the properties of being positive and symmetric.[2] From (X, k), the **affinity matrix** $W = (w_{i,j})_{(i,j)\in[1;N]^2} = (k(x_i, x_j))_{(i,j)\in[1;N]^2}$ reflects the pairwise **similarities** between the data points.

By normalizing each similarity measure $w_{i,j}$ with $d_i = \sum_{i\in X} w_{i,j}$, we obtain the **connectivity** $p_{i,j} = \frac{w_{i,j}}{d_i}$ between two data points. This quantity $p_{i,j}$ can be interpreted as the probability of moving from x_i to x_j. Unlike the similarity, it is not symmetric, yet, it has the particular characteristic that every row of $P = D^{-1}W$, where D is the diagonal matrix with its elements being $(d_i)_{i\in X}$, sums to 1.

According to the aforementioned properties, P, constructed from p(x,y), constitutes a **transition probability matrix** of a Markov chain on X. And, if $(P)_{i,j} = p(x_i, x_j)$ denotes the transition probability from x_i to x_j in one time step Δt, P^t gives the t-step transition matrix.

We can show by introducing a symmetric matrix similar to P^t for example that $P^t = \Phi\Lambda^t\Psi$, where Λ^t is the diagonal matrix containing the N eigenvalues $\lambda_0 = 1, \lambda_1, \lambda_2, ..., \lambda_{N-1}$ of P^t. Therefore, $P^t_{i,j} = \sum_l \lambda^t_l \psi_l(x_i)\phi_l(x_j)$ The diffusion map maps points from the original space to the eigenvectors of $P : X \mapsto \mathbb{R}^{n^{-1}}$, and is defined as

$$\Psi_t(x) = (\lambda^t_1\psi_1(x), \lambda^t_2\psi_2(x), \ldots, \lambda^t_{n-1}\psi_{n-1}(x))$$

Now, the eigenvalues of P all lie between 0 and 1. As time progresses and t increases, the small and intermediate eigenvalues (the ones not close to 1) quickly decay.It is then sufficient to use only the first k eigenvectors and eigenvalues. Thus, we get the diffusion map from the original data to a k-dimensional space which is embedded in the original space.

[1] A topological space M is a topological manifold of dimension d if M is locally Euclidean: each point of M has a neighborhood that is homeomorphic to an open subset of $\mathbb{R}^d$. [Lee, 2012]

[2] The Gaussian kernel is particularly popular and used. It's defined as $k(x, y) = \exp\frac{||x-y||^2}{\epsilon}$

Also, it is proved that the Euclidean distance between points in the embedded space is equal to the Diffusion Distance introduced earlier between probability distributions centered at those points, meaning Euclidean distance in diffusion space corresponds to diffusion distance in data space.

Here is a summarization of the algorithm in four steps for clarification/simplification:

1. **Affinity Matrix (W) Construction:** We first construct an affinity matrix W. The affinity between two points x_i and x_j is a measure of their similarity and is typically computed using a Gaussian kernel:

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

   where σ is a user-defined parameter controlling the width of the Gaussian kernel.

2. **Transition Probability Matrix (P) Construction:** Define the transition matrix K as a doubly-stochastic matrix, obtained by normalizing W row-wise:

$$K_{ij} = \frac{W_{ij}}{\sum_{j'} W_{ij'}}$$

   Then, define the symmetric matrix P as:

$$P = (D^{-1/2})K(D^{-1/2})$$

   where D is a diagonal matrix with elements $D_{ii} = \sum_j K_{ij}$.

3. **Eigenvalue Decomposition of the Transition Matrix (P):** Perform the eigenvalue decomposition of P to obtain the eigenvalues λ_i and corresponding eigenvectors ϕ_i:

$$P\phi_i = \lambda_i \phi_i$$

4. **Eigenvalues Selection and Diffusion Map (Y) Construction:** Choose a subset of the eigenvectors based on the corresponding eigenvalues. These eigenvectors form the columns of the matrix Φ. The diffusion map Y is then obtained by stacking the chosen eigenvectors:

$$\tilde{Y} = [\phi_1, \phi_2, \ldots, \phi_k]$$

   The choice of k is typically determined by examining the decay of the eigenvalues.

The diffusion map $\tilde{Y}$ obtained in the last step serves as a lower-dimensional representation of the original data. Points that are close in the diffusion map space are considered similar in the original high-dimensional space.

Report on task 1, Description of the datasets

Our generated dataset [4]

The Swiss roll dataset was created in 2004 to use for testing dimensionality reduction techniques. It is now a commonly used toy dataset in scikit-learn. Consisting of a set of three-dimensional points arranged in a "roll" shape, the dataset resembles the Swiss roll cake which is the origin of the dataset's name. This can be seen in Fig. 1. The points on the dataset are colored a lot of the time, making it easier to visualize the plot of the reduced data. What made the Swiss roll dataset popular is its easy generation and visualisation along with its nonlinear structure - in contrast to the linear dimensionality reduction methods, e.g. PCA.

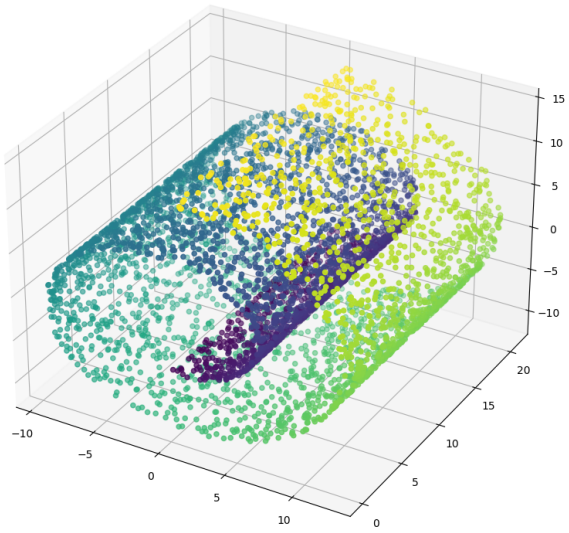


Figure 1: Swiss roll with 5000 datapoints

Word2Vec [8]

As there are many different types of similarities between words, it is sensible to use high-dimensional word vectors. Mikolov et al. discovered that simple algebraic operations can be performed using these vector representations. For example, there could be a given word pair (a, b) that holds a distinct difference. Using algebra, a word X can be found holding the same difference to a given word (c):

X = a - b + c

For example (see Fig. 2):
vector(”king”) = vector(”man”) - vector(”woman”) + vector(”queen”)
vector(”smallest”) = vector(”biggest”) - vector(”big”) + vector(”small”)

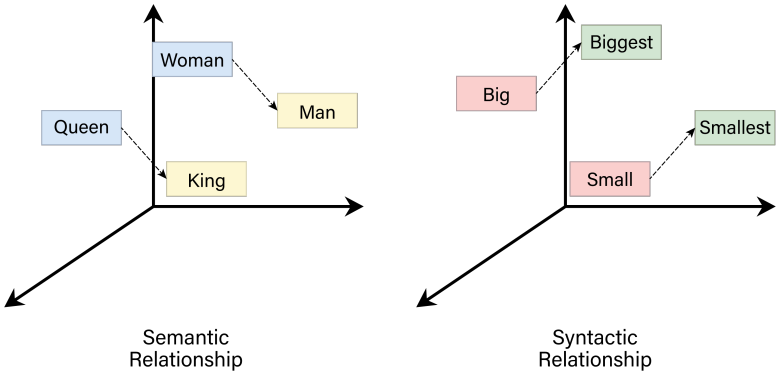


Figure 2: Semantic and Syntactic Relationships in Word2Vec

When high-dimensional word vectors are trained on a large amount of data, very subtle semantic relationships between words can be discovered. Next to semantic similarities, the team of Google also looked at the words in the context of syntactic similarities. Several tests were run on different simple model architectures and compared on accuracy. As a result of one test, the team realized that using multiple examples instead of just one to form the relationship vector improved the accuracy of the best models by about 10% absolutely on the semantic-syntactic test. The models were all implemented in a distributed framework titled DistBelief; they were trained on the Google News 6B data set, with mini-batch asynchronous gradient descent and Adagrad. After the first publication of the paper, Mikolov et al. published code for computing the word vectors with an improved training speed as well as more than 1.4 million vectors of dimensionality 300 representing named

entities, trained on more than 100 billion words. The Word2Vec dataset now is one of the most popular implementations of word embedding and has been commonly used throughout its existing years.

Galaxy spectra [3] [13]
Next to Word2Vec, Megaman also used galaxy spectra from the Sloan Digital Sky Survey (SDSS). While it is not specified in the paper, according to the year of publication, the data is likely to be from the SDSS-III release.

SDSS made it their mission to observe stars, galaxies, and quasars and provide the world with near-continuos data. So far, there have been five phases. The first one, SDSS-I, dates back to 2000. Among other things, optical spectra of more than 700,000 objects were collected. SDSS-III was published in 2011 and put its focus on spectroscopy. Their survey required an upgrade to optical spectrographs. Additionally, infrared spectrographs were introduced.

The spectrum of a galaxy represents the emitted or absorbed light across a range of wavelengths from the various objects within that galaxy. It provides information about the composition, temperature, density, and motion of the contributing astronomical entities within the galaxy. The spectrum is a valuable tool for astronomers to study the characteristics and properties of galaxies.

In the Megaman paper, an article is referred to view the preprocessing of this galaxy spectra data. However, it is stated to be "in preparation" and we could not find a publication. Therefore, we decided to focus on the Word2Vec dataset used in the paper.

Report on task 2, Implementation of the SpectralEmbedding method

In this exercise, we will handle multidimensional and large datasets, the details of which will be elaborated upon. To address the challenge of dimensionality reduction, we will employ a method known as Spectral Embedding. This technique is particularly effective for complex, non-linear data structures, as it utilizes the spectral properties of data. In fact, Spectral Embedding is similar to Diffusion Maps formerly explained and finds applications in clustering, visualization, and dimensionality reduction tasks. Spectral Embedding also works by constructing a matrix to represent the affinities or relationships between data points, followed by an analysis of the matrix's spectrum, specifically its eigenvalues and eigenvectors, to achieve dimensionality reduction. However, compared to Diffusion Maps, the emphasis on different eigenvectors (we might choose eigenvectors corresponding to the smallest eigenvalues, which capture the global structure of the data, rather than the fastest decaying eigenvalues as in Diffusion Maps) may lead to subtle differences in the resulting embeddings and their interpretability.
This method is supported by various Python libraries dedicated to manifold learning, including `scikit-learn` [9], `Megaman` [7], and `Datafold` [5]. Our implementation will focus on `Datafold` for Spectral Embedding, which will then be compared with the implementation in `Megaman`.

Megaman is a scalable manifold learning package developed in Python, designed particularly for large data sets. Megaman's API is designed to be familiar to users of scikit-learn, inheriting much of its functionality. This makes it easier for those already accustomed to scikit-learn to adapt to Megaman. It requires standard Python libraries like NumPy, SciPy, and scikit-learn to function. Megaman utilizes the C++ Fast Library for Approximate Nearest Neighbors (FLANN) and the Sparse Symmetric Positive Definite (SSPD) solver Locally Optimal Block Preconditioned Gradient (LOBPCG) method. An overview of the classes and packages used can be seen in Fig. 3. The self-implemented classes are depicted in gray, framed, while packages are in gray, with no frame, and external packages in blue.

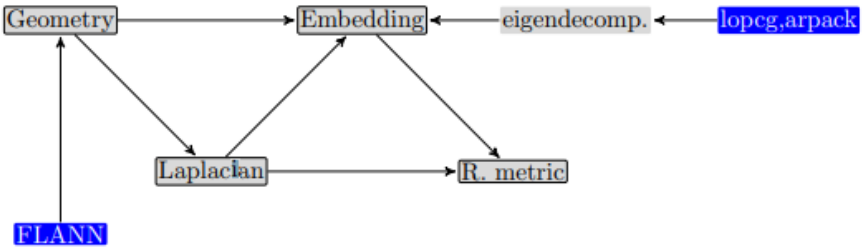


Figure 3: Understanding Megaman

These advanced tools help in scaling manifold learning algorithms for handling large datasets efficiently. Megaman is specifically designed for high performance and research purposes. Megaman caches intermediary steps and indices, facilitating fast re-computation with new parameters, a valuable feature for experimental and iterative research work. The package implements several manifold learning algorithms in its own classes, which inherit from a base class. These include `SpectralEmbedding`, LTSA, LocallyLinearEmbedding, and Isomap. Each of these algorithms has its own significance and applications in manifold learning. We are going to focus on `SpectralEmbedding`, which, as stated earlier in the paper, is a manifold learning technique that typically deals with the graph structure of the data, focusing on capturing the essence of the data in fewer dimensions by using the eigenvectors of the Laplacian matrix. The matrix's eigenvalues and eigenvectors are then computed, with the eigenvectors corresponding to the smallest non-zero eigenvalues serving as the new data representation. This preserves local pairwise distances, making it particularly effective for data that lies on a curved manifold. Spectral embedding focuses on the graph structure directly without simulating a diffusion process over time.

The Geometry class in Megaman implements geometric operations common to many embedding algorithms, such as computing distances and Laplacians. This class enhances the versatility and applicability of the package in manifold learning tasks. The RiemannianMetric feature in Megaman estimates the Riemannian metric, enhancing the precision of manifold learning. Moreover, the eigendecomposition module provides a unified interface to various eigendecomposition methods available in SciPy, streamlining the process of implementing these methods.

We will compare the spectral embedding method with `Datafold`'s diffusion maps. Datafold is a Python package that provides data-driven models for point clouds to find an explicit manifold parametrization and to identify non-linear dynamical systems on these manifolds. The explicit data manifold treatment allows prior knowledge of a system and its problem-specific domain to be included. Especially, Datafold utilizes diffusion maps for non-linear dimension reduction to reveal geometric structures within data. This technique defines a diffusion process on data, simulating how points interact over time, and employs eigenfunctions to capture these dynamics. The alpha parameter adjusts the approach, aligning it with mathematical concepts like the Graph Laplacian. Datafold's implementation, found in the `datafold.dynfold.DiffusionMaps` class, supports diverse data formats and includes features like inverse transformations for mapping from the reduced dimensional space back to the original space. Diffusion maps, as a manifold learning strategy, are part of the spectral methods family but are unique in their focus on data's diffusion properties.

That's why we decided to compare the different libraries and algorithms.

Report on task 3, Description of the libraries used

- **Scikit-learn**
  Scikit-learn, often abbreviated as Sklearn, includes a sub-module known as 'manifold', which is widely recognized for offering a comprehensive open-source implementation of various manifold learning algorithms. This Python package is primarily focused on user-friendliness. Although Scikit-learn includes algorithms aimed at scalability, its manifold learning techniques do not fall into this category. The methods provided in the manifold sub-module are not capable of processing out-of-core data, making it challenging to scale these methods for large datasets. Additionally, these methods face limitations in sharing intermediate results, such as eigenvector computations, between different algorithms. This can lead to inefficiencies during the data exploration process. To address these limitations, the Megaman package was developed.

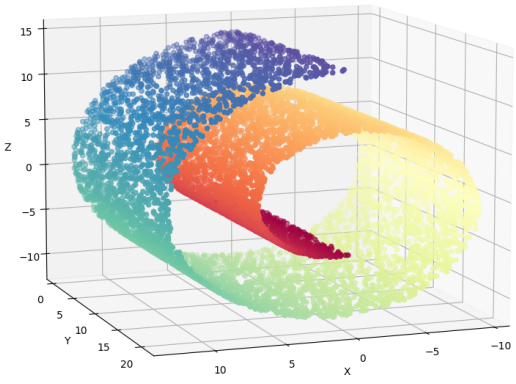
In our study, we utilized the Spectral Embedding functionality from the scikit-learn package to conduct experiments on 2 datasets in different conditions, which are elaborated upon as follows.

– **Sklearn Spectral Embedding on Swiss Roll dataset (10k points):**
In our exploration of dimensionality reduction techniques using the scikit-learn package, we encountered a formidable obstacle due to the sheer volume of our data. Initially, we generated a Swiss roll dataset with 100,000 samples, aiming to apply Spectral Embedding, a sophisticated method for reducing data dimensions. Aware of the computational challenges posed by such a large dataset, we initially considered minibatch learning as a potential solution. Minibatch learning is an effective strategy for handling vast datasets by dividing them into smaller, more manageable batches. This method proves invaluable when dealing with datasets too large for the available computer memory, facilitating efficient use of limited memory resources.

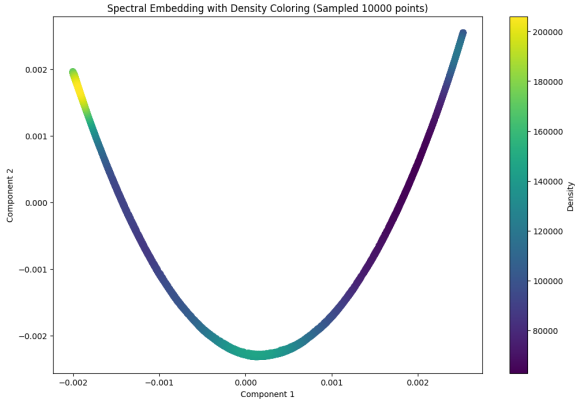
However, the nature of Spectral Embedding necessitates a different approach. Unlike methods that can be adapted to minibatch learning, Spectral Embedding requires a comprehensive analysis of the dataset's overall structure. It computes eigenvectors and eigenvalues from the Laplacian matrix, capturing the connectivity and similarities among data points. This computation demands simultaneous access to the entire dataset to ensure accurate embedding outcomes, as it hinges on a holistic understanding of the data's global interactions.

To address the computational demands of processing the full dataset of 100,000 points with standard hardware, such as a computer with 12.7GB of RAM, we introduced an affinity condition based on the concept of nearest neighbors. We opted to limit our affinity calculations to the 40 nearest neighbors for each data point, a decision aimed at managing computational load while preserving the integrity of local neighborhoods essential for effective Spectral Embedding. This approach significantly reduces computational requirements by focusing on local structures, thereby making the process feasible on normal hardware without sacrificing the quality of the dimensionality reduction.

Applying the 40-nearest-neighbors condition uniformly across analyses of both the 10,000- and 100,000-point datasets, we endeavored to ensure consistency in our experimental setup. This strategy was crucial for effectively reducing the computational burden, focusing on local point interactions, and excluding less relevant distant points. It allowed us to conduct our analysis within the memory constraints of standard computing resources while maintaining the local neighborhood's integrity, which is critical for the successful application of Spectral Embedding on large datasets. This approach underscores the challenges and innovative solutions required when dealing with large-scale data in the realm of dimensionality reduction.



(a) Swiss roll sampled 10k points



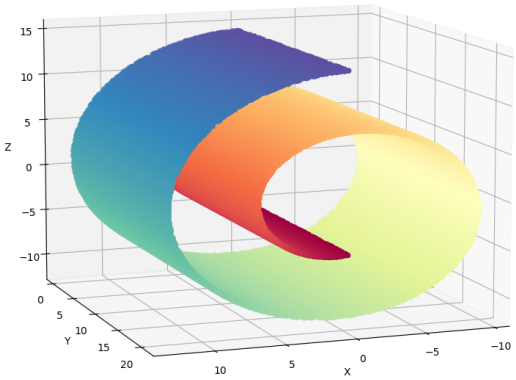
(b) Spectral Embedding with density coloring Visualization of 10k point

Figure 4

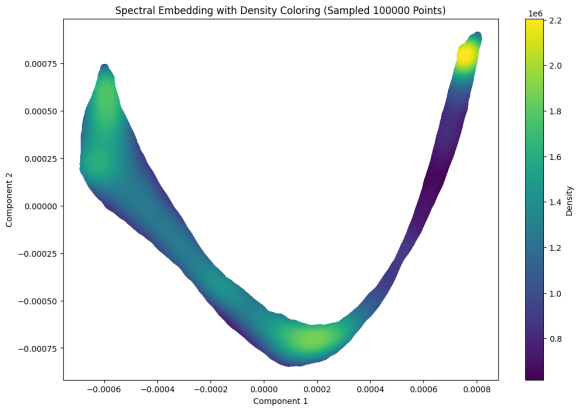
The scatter plot generated in Fig. 4 illustrates the application of Spectral Embedding on the Swiss roll data, with the aim of transforming it into a two-dimensional space. The primary objective of

Spectral Embedding is to unfold the Swiss roll's manifold into a planar, two-dimensional representation, all the while maintaining the proximity of each point to its neighbors. The color scheme utilized in the plot mirrors that of the original three-dimensional visualization, denoting the original longitudinal position of each point on the manifold. The expectation from Spectral Embedding is the preservation of the color gradient in the 2D projection, which signifies that points adjacent to the manifold should ideally remain close in the reduced-dimensional space. However, the observed outcome did not align with this expectation, as the color gradient preservation—and consequently, the local neighborhood structure—was not as distinct as anticipated. This observation was further substantiated by a relatively low trustworthiness score of 0.4993, indicating that the embedding may not have effectively maintained the manifold's intrinsic geometric properties.

– **Sklearn Spectral Embedding on Swiss Roll dataset (100k points):**
We expanded our dataset to include 100,000 points, as was explained above, presenting a significant challenge due to its size. Seen in subfigure 5b, the 'V' shaped plot indicates that the manifold has been unfolded. This representation is in a two-dimensional space where the x- and y-axes correspond to the two dimensions obtained from Spectral Embedding. The juxtaposition of these plots serves to illustrate the process of dimensionality reduction and how it can be visualized both before and after the application of Spectral Embedding. Also, it comes with density coloring which is especially useful for understanding how the points are distributed in the reduced space, offering insights into the underlying structure of the dataset that may not be immediately apparent from the raw Spectral Embedding alone.



(a) Swiss roll sampled 100k points



(b) Spectral Embedding with density coloring Visualization of 10k point

Figure 5

– **Sklearn Spectral Embedding on Google News Word2Vec dataset (10k points):**
We applied spectral embedding to real-world data using the Word2Vec model, which was pre-trained on Google News. This model represents words in its vocabulary as 300-dimensional vectors. For our analysis, we randomly selected a subset of 10,000 word vectors from the Word2Vec model, along with their corresponding vectors.

The considerable size of the dataset posed a challenge, as it was too large to be processed in one fell swoop given memory limitations. To circumvent this, we turned to batch processing — an approach that, while not ideal, was chosen over other dimensionality reduction techniques like PCA due to its suitability to our specific constraints. This method involved dividing the dataset into manageable portions. For the 10,000 vector sample, we processed in batches of 1,000, which allowed us to perform the spectral embedding incrementally. When handling the more extensive 100,000 vector sample, we scaled the batch size up to 10,000. This strategic partitioning enabled the efficient transformation of high-dimensional word vectors into a lower-dimensional representation, ensuring the process remained manageable within the confines of our hardware's memory capabilities.

The data is embedded into 2 dimensions, and the result in Fig. 6 shows a dense clustering of points

in the center, suggesting that many word vectors are mapped close to each other in this reduced space. This could mean that a large number of words have similar contexts within the training corpus, leading to similar word vectors. Also, there are no clear, distinct clusters visible in the plot. In an ideal scenario, we might expect to see clusters of points that correspond to semantically similar words. The absence of such clusters might indicate that the Spectral Embedding has not effectively separated the words into distinct groupings based on their semantic relationships. It's also possible that the original high-dimensional space was very dense, and the Spectral Embedding is reflecting that density.

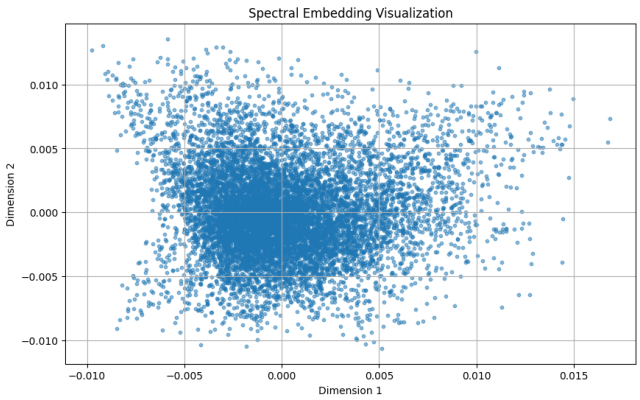


Figure 6: Spectral Embedding on Word2vec 10k points

– **Sklearn Spectral Embedding on Google news Word2Vec dataset (100k points):**
With a larger sample size, the visualization is more crowded, and the spread of the points is tighter. This could suggest a more accurate representation of the true relationships between words in the high-dimensional space, as more data points can lead to a better estimation of the true structure of the data.

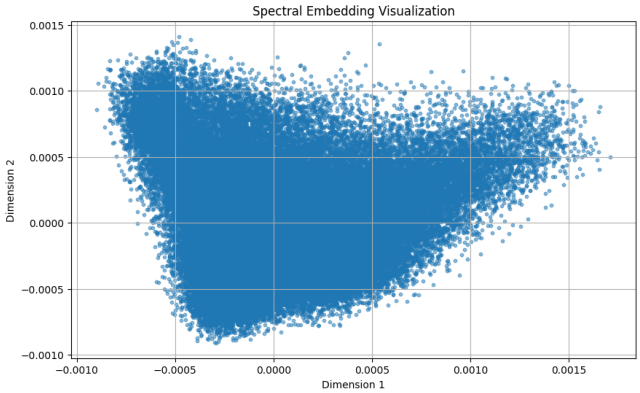


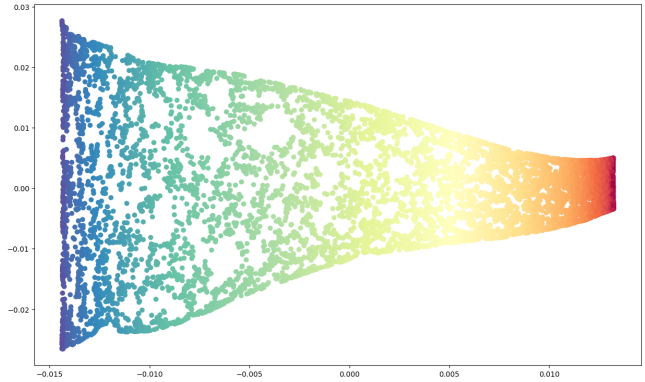
Figure 7: Spectral Embedding on Word2vec 100k points

The resulting visualization for both samples seems to show a distribution of points in a dense, central band with fewer points as it radiates outward. This could indicate that many of the word vectors are similar to each other, forming a dense cluster in the center, while more unique or less frequent words are represented by points further away from the center.

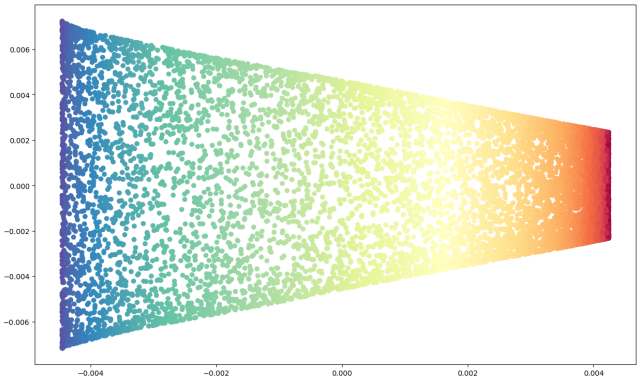
- **Datafold Library**
One of the libraries that particularly helped us learn embeddings effectively was the datafold library. It is a Python package that provides data-driven models for point clouds to find an explicit manifold parametrization and to identify non-linear dynamical systems on these manifolds. Datafold provides us with an efficient implementation of the Diffusion Maps algorithm which we can use to learn embeddings

on higher dimensional with large sizes too. The scalability factor provided by the datafold makes it a particularly useful choice for us to use on big datasets. We carried out multiple experiments with the library on our datasets which are discussed below.[2]

– **Datafold Diffusion maps for 10k points on Swiss Roll dataset:**
We start with a simple implementation of the diffusion maps on a moderately sized Swiss roll curve dataset consisting of 10k points. We first generate the dataset using sklearn's *make_swiss_roll* function. One of the biggest challenges in applying diffusion maps is finding the optimal parameters. But, this becomes easy with datafold as it provides a function to find optimal parameters. We use the PCManifold *optimize_parameters* function to get the optimum value of the ϵ and *cutoff* values. Next, we use these parameters to fit on the Swiss roll curve dataset and compare potential two-dimensional embeddings. We can visually compare which eigenvectors provide us with the most information or in other words do the best job of unfolding the Swiss role curve into two-dimensional embedding space by plotting the first non-trivial eigenvector with other computed eigenvectors. Datafold also provides us with the option of automatic selection of embeddings via the *LocalRegressionSelection* model. It tells us that the eigenvectors 1 and 5 are the best choice which can be seen in the figure 8a. We also get a very high trustworthiness score for our dataset which represents high quality embeddings.



(a) 10k points



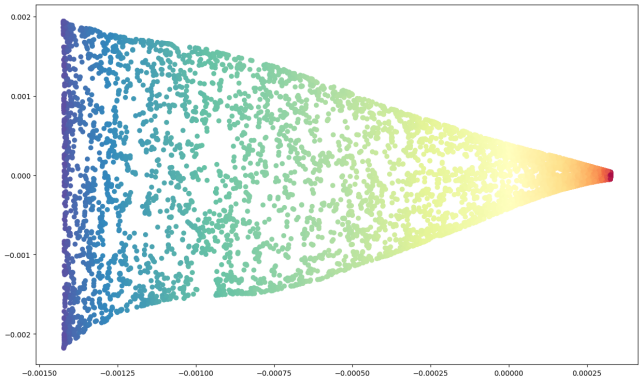
(b) 100k points

Figure 8: Datafold diffusion map results on Swiss roll dataset

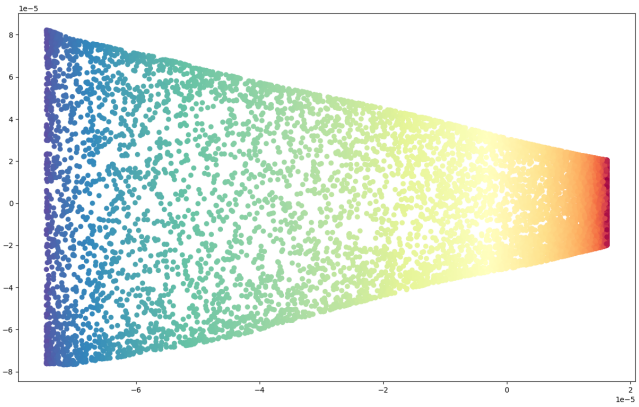
– **Datafold Diffusion maps for 100k points on Swiss Roll dataset:**
Building upon our initial experiment, we escalated the complexity by increasing the number of points in the Swiss Roll Curve to 100,000. This expansion was aimed for us to test the scalability and robustness of the Datafold's diffusion maps algorithm. We mirrored the steps that we took for 10,000 points and fitted the diffusion maps models using the optimal parameters given by the PCManifold *optimize_parameters* function. The results can be seen in figure 8b. The trustworthiness score was also very high representing good embeddings.

Naturally, it took more time to compute the embeddings, but datafold was able to find it efficiently and quickly on our not so powerful computers while other libraries like Sklearn failed to do so without minibatching which might not even give particularly good results as discussed above.

– **Datafold's Scalable Embedding via Landmark Diffusion on Swiss Roll dataset for 100k points:**
We get satisfactory results with the original implementation of datafold's diffusion map algorithm but it also exhibits a notable trade-off with computational time, especially when we increase the dataset size. It is also unable to deal with very large datasets: in our case, it couldn't handle a Swiss Roll curve with 1 million points. Fortunately, datafold offers us an alternative method that not only accelerates the embedding generation process but also extends the compatibility to substantially larger datasets.
This is done via the Roseland algorithm which can be viewed as a generalization of the diffusion map algorithm sharing various properties. The Roseland algorithm utilizes a "landmark set" which can be much smaller than the full dataset and employs this to calculate the affinity matrix. In particular, when constructing the transition probability matrix, which can be especially computationally intensive for large datasets, the Roseland algorithm doesn't use all of the data points but only a subset (which we referred to as the "landmark set"). This allows to reduce the computational complexity with the classical Diffusion Maps algorithm from $O(N^3)$ to $O(n^2N)$ or $O(n^3)$, where N is the total number of data points and n is the number of selected points for approximation. In the end, the datafold's implementation of the Roseland algorithm allows us to get faster results on the Swiss roll dataset with 100k points with comparable trustworthiness metrics while providing much faster results.
The steps involved in applying the Roseland algorithms are mostly the same with just an extra choice of selecting the landmark parameter. We tried different landmark parameters and found that a value of 0.4 gave pretty good results without compromising the computational time. While the diffusion map algorithm took somewhere close to 7 minutes to calculate the embeddings, the Roseland algorithm calculated in around 1 minute which is a big improvement in runtime. The results obtained by the Roseland algorithm can be seen in figure 9a



(a) 100k points



(b) 1 million points

Figure 9: Datafold Scalable Embedding via Landmark Diffusion

– **Datafold's Scalable Embedding via Landmark Diffusion on Swiss Roll dataset for 1M points:** Extending on our previous implementation using the Roseland algorithm[11], we finally tried testing the library on 1 million points which has not been possible using the other libraries like sklearn and UMAP. But, to our surprise, with a landmark value of 0.2, we were able to generate the 2-dimensional embedding of the Swiss roll curve using Datafold's Roseland model. Although choosing a lower landmark value forces us to tradeoff between accuracy and computational runtime, it allows us to get meaningful results on our not so powerful computers. This is particularly exciting and is comparable to the Megaman package itself which allowed us to use the Spectral Embedding method on very large datasets. The result of the embeddings can be seen in the figure 9b. Although it took us more than an hour and a half to calculate the embeddings, the possibility of calculating it is itself a big advantage.

– **Datafold Diffusion Maps on Google news Word2Vec dataset (10k points):** We also applied the original Diffusion Map model provided by datafold on the Word2Vec dataset. In order to run the algorithm, we had to tweak the parameters ϵ to a pretty low value of 0.1 as running it with a higher epsilon value or the optimal value suggested led to a memory error due to extensive internal calculations. Although we were able to apply the Diffusion Maps model, we couldn't get satisfactory results from it as can be seen in figure 10.
Several factors might be responsible for this outcome. First, there could be some issues with the pre-processing of the dataset, and it might need some refinement to improve the algorithm's performance on this specific dataset. Alternatively, the Diffusion Map algorithm may not be the best choice for dimensionality reduction in the context of the Word2Vec dataset. We might need to look at other algorithms capable of generating the embeddings for this dataset.

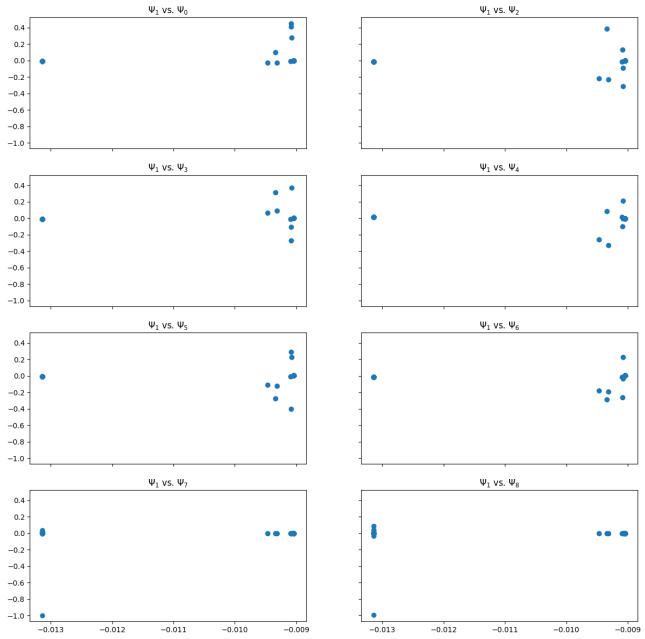


Figure 10: Diffusion Map on Word2Vec

- **UMAP and NVIDIA's RAPIDS:**

  In our quest for high-speed data analysis, we discovered the power of Uniform Manifold Approximation and Projection (UMAP), a dimensionality reduction technique introduced in 2018 by McInnes et al.[6] UMAP stands out due to its ability to preserve both local and global structures within data, providing a comprehensive representation of the original high-dimensional space. This is a significant improvement over techniques such as t-SNE (t-Distributed Stochastic Neighbor Embedding), which primarily focuses on local structure preservation.

  UMAP operates by leveraging the concept of a manifold, a shape that resembles a flat Euclidean space when zoomed in. The algorithm constructs a weighted graph from the high-dimensional data, where each data point is a node, and edges are drawn between nodes that are close to each other in the high-dimensional space. The weight of each edge corresponds to the distance between the nodes it connects. This graph construction is based on the concept of a fuzzy simplicial set, a mathematical structure used to capture the topological features of the manifold.

  Once the graph is constructed, UMAP seeks to find a lower-dimensional representation of the data that preserves the topological structure of the original high-dimensional data as much as possible. This is achieved by minimizing a function that measures the difference between the high-dimensional graph and the lower-dimensional representation using a variant of stochastic gradient descent. The result is a lower-dimensional representation of the data that maintains the essential topological features of the original high-dimensional data, making UMAP a powerful tool for visualizing and understanding complex datasets.

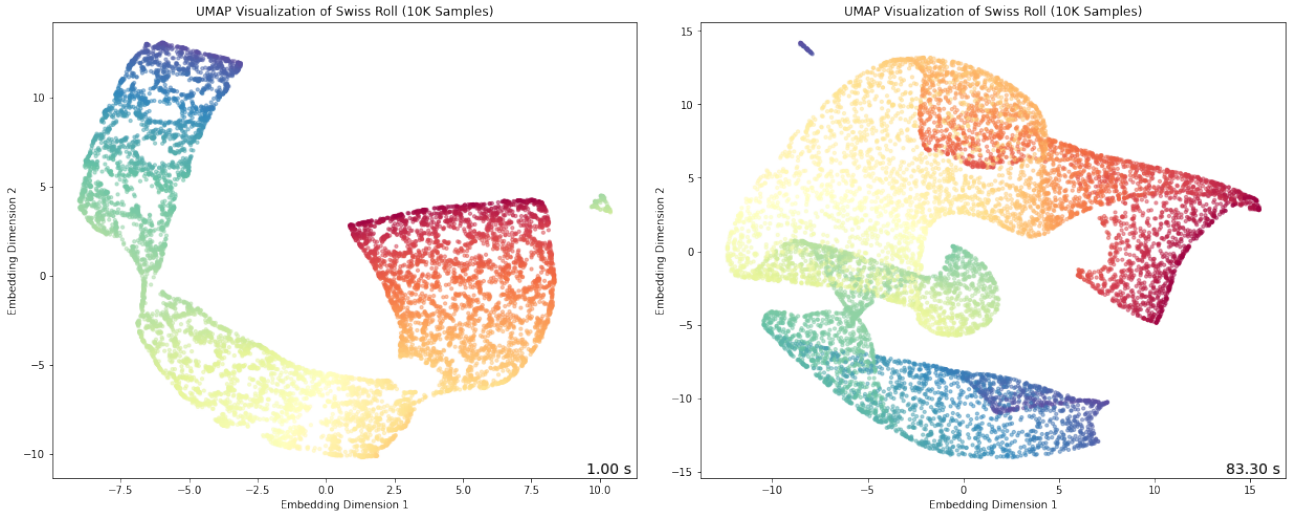
  To further enhance the efficiency of UMAP, we turned to NVIDIA's RAPIDS library, an open-source suite of GPU-accelerated data science and AI libraries. Specifically, we utilized its cuML component, which provides a GPU-accelerated implementation of UMAP. This implementation leverages the parallel processing capabilities of the GPU, operating on a random sample of the full dataset for enhanced efficiency with larger datasets.

  Built on NVIDIA CUDA-X AI, RAPIDS simplifies the complexities of GPU operations and data center communication protocols. It offers the flexibility to run anywhere—cloud or on-premises—and can scale from a workstation to multi-GPU servers to multi-node clusters. This combination of UMAP and NVIDIA's RAPIDS library presents a powerful and efficient solution for high-speed data analysis.

  Experiments using Rapids' UMAP

In our study, we conducted tests on two datasets: sklearn's Swiss roll and a dataset provided by Google. Both datasets contain 10,000 and 100,000 data points. Despite our efforts to manage GPU memory efficiently, we encountered a bottleneck in memory capacity when attempting to increase the number of data points significantly. This limitation suggests that, despite its many advantages, UMAP may not be the optimal solution for our problem when dealing with extremely large datasets. Further research is needed to explore alternative methods or optimizations that can overcome this memory constraint. This finding underscores the importance of considering the specific requirements and constraints of a problem when selecting a dimensionality reduction technique.

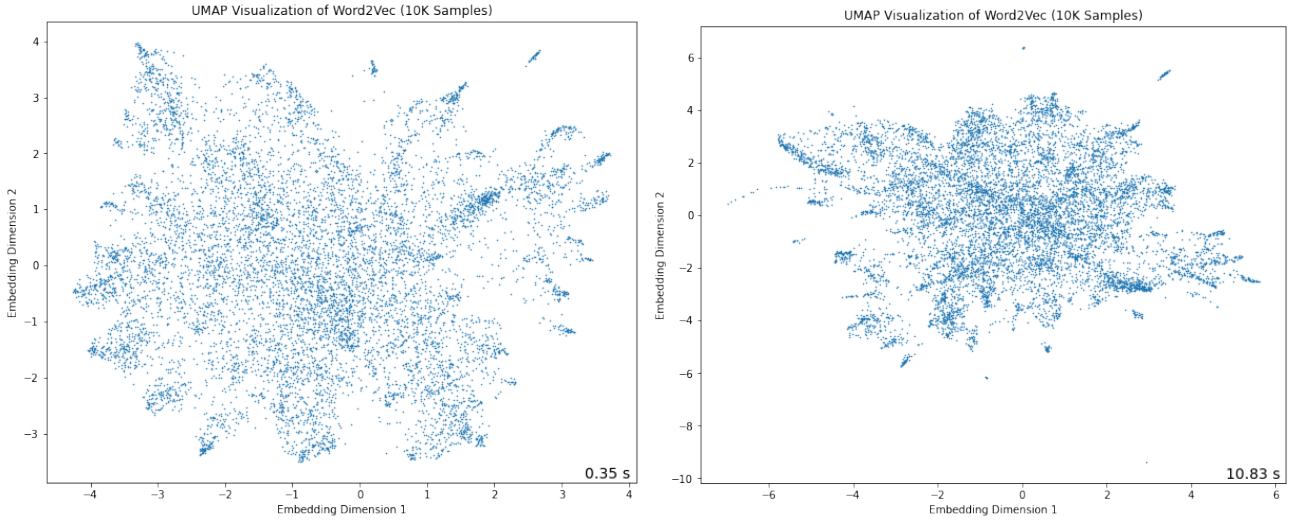
- **Dimensionality reduction on swiss roll dataset (10k points)** Our initial tests were conducted on a Swiss roll dataset comprising 10,000 points. As anticipated for a dataset of this size, the UMAP algorithm was executed swiftly. The accuracy of the dimensionality reduction was high, as evidenced by a trustworthiness rate of 0.9997. This result underscores the efficacy of UMAP in accurately reducing dimensions while maintaining the integrity of the data structure. [Figure 11a]
- **Dimensionality reduction on swiss roll dataset (100k points)** To further evaluate the capabilities of UMAP, we expanded the dataset to 100,000 points. While there was a slight decrease in accuracy, the trustworthiness rate remained high, indicating that UMAP effectively handles larger datasets with minimal impact on performance. [Figure 11b]



(a) 10K DataPoints (b) 100K DataPoints (only random 10k shown)

Figure 11: UMAP SwissRoll

- **Dimensionality reduction on Google news Word2Vec dataset:** Our exploration extended to Google's Word2Vec dataset, with tests conducted on both 10,000 [Figure 12a] and 100,000 [Figure 12b] data points. Interestingly, we observed a notable decrease in accuracy compared to the Swiss roll dataset for both cases. However, this reduction in accuracy was accompanied by an increase in computational speed. This suggests that while UMAP's performance on the Word2Vec dataset was faster, it came at the cost of accuracy. This trade-off between speed and accuracy is a crucial aspect to consider in the context of large-scale, real-world applications.

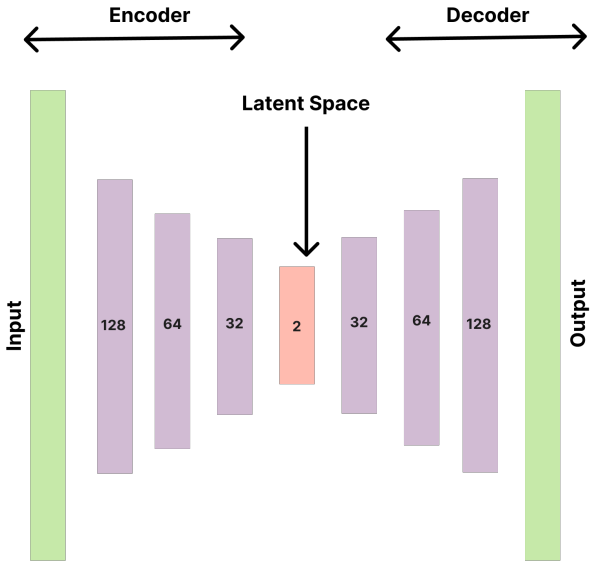


(a) 10K DataPoints (b) 100K DataPoints (only random 10k shown)

Figure 12: UMAP Word2Vec

- **Autoencoder based dimensionality reduction**
  We also used an innovative approach utilising an autoencoder to reduce the dimensionality of our input features, yielding lower-dimensional embeddings on our datasets.
  Autoencoders prove to be particularly advantageous in our context, excelling at capturing intricate non-linear relationships with the data. Throughout the training phase, the autoencoder dynamically learns to encode and decode the input features by establishing a compact latent representation. This lower-dimensional latent representation aims to capture the most salient and informative features of our datasets. After the completion of the training, we can extract the encoder component of the autoencoder. This extracted encoder serves as a dimensionality reduction tool providing us with reduced dimensional data that captures the most essential information of the input data.

  Autoencoder architecture
  A simplified representation of our autoencoder architecture can be seen in figure 13a. It consists of multiple dense layers with varying numbers of neurons. The input layer receives the data. Following the input layer, we have a dense layer with 128 neurons, connected to a layer with 64 neurons and subsequently to a layer with 32 neurons. This final layer produces a 2-dimensional encoded representation, effectively capturing the essential features of our input data.
  The decoder follows a mirrored architecture to reconstruct the original input from the latent representation. We also used Batch Normalization in the model to facilitate a smoother learning process. A comprehensive overview of the model structure can be seen in figure 13b.



(a) Autoencoder architecture

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 3)]               0

 batch_normalization (Batch  (None, 3)                 12
 Normalization)

 dense (Dense)               (None, 128)               512

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 2)                 66

 batch_normalization_1 (Bat  (None, 2)                 8
 chNormalization)

 dense_4 (Dense)             (None, 32)                96

 dense_5 (Dense)             (None, 64)                2112

 dense_6 (Dense)             (None, 128)               8320

 dense_7 (Dense)             (None, 3)                 387

 batch_normalization_2 (Bat  (None, 3)                 12
 chNormalization)

=================================================================
Total params: 21861 (85.39 KB)
Trainable params: 21845 (85.33 KB)
Non-trainable params: 16 (64.00 Byte)
_________________________________________________________________
```

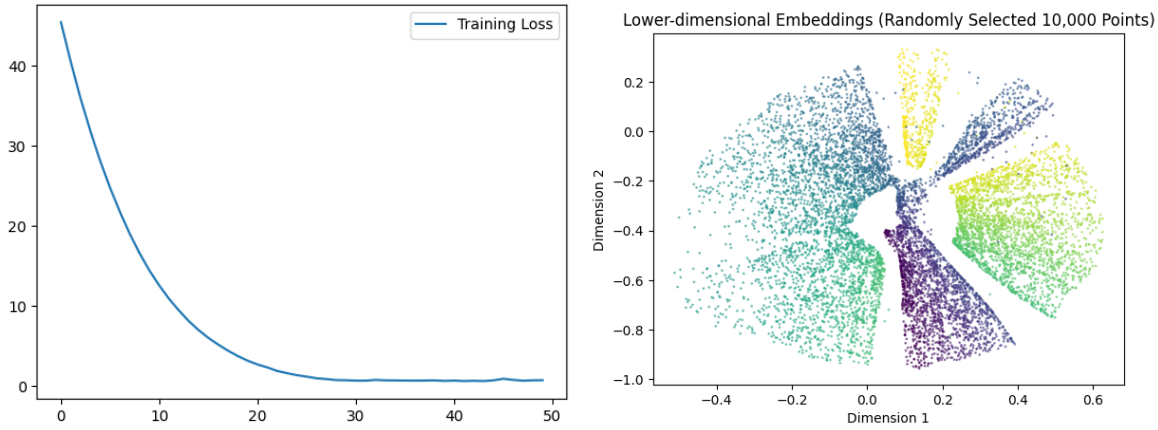
(b) Autoencoder summary

Figure 13: Autoencoder structure

Experiments with the autoencoders:

– **Dimensionality reduction on swiss roll dataset (100k points):**
We again utilized the sklearn's *make_swiss_roll* function to create a Swiss roll dataset with 100,000 points. We then used this dataset to train our autoencoder. Since our main goal was to understand the hidden patterns in the data and learn a lower dimensional embedding, we skipped the usual step of dividing the dataset into parts to check the model's performance. We wanted our model to get very good at understanding the specifics of this data, so we let it focus only on learning from it. This straightforward approach helped the model to get better at capturing the unique features of the swiss roll dataset.
We trained this model for 50 epochs without GPU acceleration to mimic a weak computer and to our surprise, we were able to train this model on 100k points in around 10 minutes which can be

compared to the datafold's performance on the same dataset size. The loss curve can be seen in figure 14a. We plotted the results for randomly selected 10,000 points for easy visualization as can be seen in figure 14b. The results clearly show some patterns emerging which might be useful in understanding the dynamics of the underlying curve of the swiss roll dataset. The trustworthiness metric is also comparable with the score achieved using datafold.

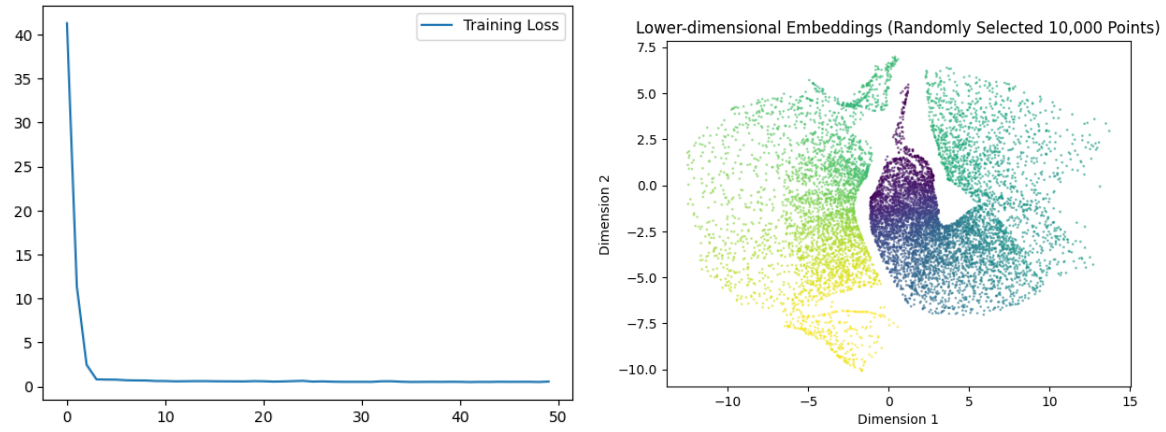


(a) Training Loss curve

(b) Embedding result

Figure 14: Autoencoder training curve and result on Swiss roll curve for 100,000 points.

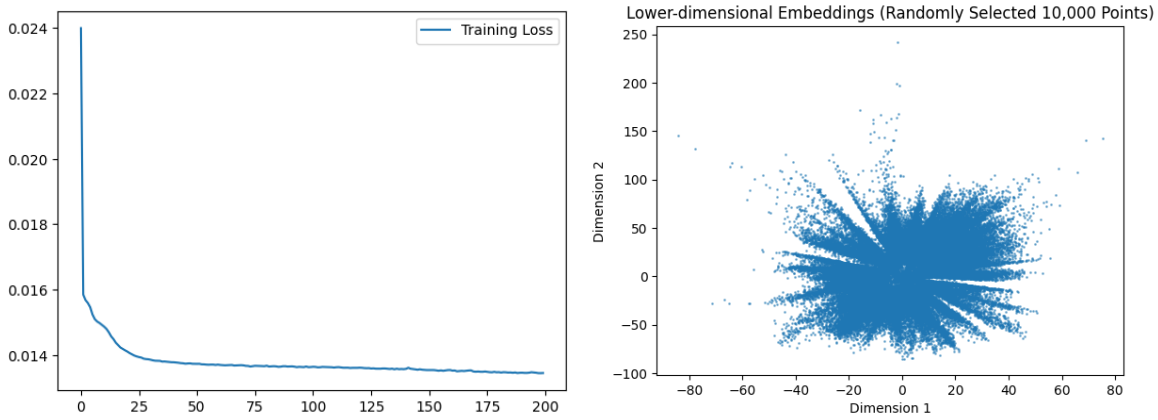
– **Dimensionality reduction on Swiss roll dataset (1 million points):**
We replicated the steps that we used in our previous implementation of the autoencoder on a dataset with 100,00 points, extending our approach to train the autoencoder on an even larger dataset comprising 1 million points. This poses a significant challenge for other libraries and algorithms including Diffusion Maps.
These algorithms often struggle with scalability when dealing with very large datasets as they require extensive computations and memory resources. They face difficulties in efficiently preprocessing and extracting meaningful embeddings from datasets with a million points.
In contrast, our autoencoder implementation showcased robust performance, successfully learning embeddings of the Swiss roll dataset with 1 million points. The autoencoder uses minibatching during its learning process to look at the data and learn the embeddings. This might be problematic as it might not retain the global relationships within in the data. But, even with these limitations, we are able to learn embeddings on the dataset with a good local trustworthiness metric. Naturally, training the autoencoder on such huge datasets takes time. The loss curve can be seen in figure 15a. The resulting embeddings can be seen in figure 15b.



(a) Training Loss curve (b) Embedding result

Figure 15: Autoencoder training curve and result on swiss roll curve for 1 million points.

– **Dimensionality reduction on Google news Word2Vec dataset:**
Having obtained good results on the Swiss Roll Dataset, we decided to use our autoencoder to learn the embeddings on a real-life dataset of Google News Word2Vec dataset consisting of 3 million points each characterized by 300-dimensional features.
To test the efficacy of our model, we only picked a random sample of 100,000 data points and tried to fit our model on this dataset. The training loss curve and results can be seen in figure 16. Although we were able to fit our model on the dataset, the results as seen in Fig. 16b are not satisfactory and do not give proper embeddings with only a trustworthiness score of 64%.
Several factors could be responsible for the subpar performance of our model on this dataset. One potential reason may be insufficient preprocessing of the dataset. Additionally, the complexity and dimensionality of the Google News Word2Vec dataset pose a unique challenge. Inadequate tuning of hyperparameters or model architecture for such high-dimensional data can also hinder the model's capacity to capture relevant patterns effectively.
Further investigation into the specific characteristics of the dataset, experimentation with different preprocessing techniques, and fine-tuning of model parameters may be necessary to improve the performance on this challenging real-world dataset.



(a) Training Loss curve (b) Embedding result

Figure 16: Autoencoder training curve and result on Google News Word2Vec dataset (100,000 points)

Report on task 4, Metrics and Comparison

We considered different metrics to use for comparison of our results to the ones specified in the paper. Since we cannot precisely recreate the same level of accuracy, comparing the runtime to the one in the paper is not an option. Instead, we took a look at the Riemannian metric as well as trustworthiness.

Riemannian metric

"A Riemannian metric g is a symmetric and positive definite tensor field which defines an inner product $<,>_g$ on the tangent space $T_p\mathcal{M}$ for every $p \in \mathcal{M}$." [10]

It is used to define distances and angles on a smooth manifold, which is a generalization of surfaces to higher dimensions. Mathematically, a Riemannian metric on a manifold M is often denoted by a positive definite symmetric bilinear form g, defined on the tangent space at each point:

$g_p : T_pM \times T_pM \to \mathbb{R}$

Here, T_pM is the tangent space at a point p on the manifold, and g_p is the Riemannian metric at that point. The Riemannian metric allows for the calculation of lengths of curves, angles between curves, and various other geometric properties of the manifold.

We ultimately decided to use trustworthiness which is an easy concept to grasp. Additionally, it has the advantage that we can use one metric for all algorithms used in our project.

Trustworthiness [12]

Trustworthiness is a score that displays to what extent the local structure is maintained. It is always a floating point number within [0, 1] and can be computed as follows:

$$T(k) = 1 - \frac{2}{nk(2n - 3k - 1)} \sum_{i=1}^{n} \sum_{j \in \mathcal{N}_i^k} max(0, (r(i, j) - k)) \tag{1}$$

For each sample i, its k nearest neighbors in the output space are denoted as $\mathcal{N}_i^k$, and every sample j is its $r(i, j)$-th nearest neighbor in the input space. This means that any unanticipated nearest neighbors in the output space are penalised in proportion to their rank in the input space.

This metric needs to calculate the distance matrix for each point in the dataset which becomes a problem as our memory is not big enough to store this calculated matrix. In order to get a general idea of trustworthiness, we employed mini batching. We picked a subset of 10,000 vectors and calculated the trustworthiness of this smaller subset and finally calculated the mean of all of the subsets to find an approximate trustworthiness metric. This approach may not be very suitable as we need to look at the entire dataset to calculate our metric, but it gives us an approximate value to compare.

Timings

We can also compare the runtime between different libraries by utilizing one standard computer and recording the runtime. Although this is not a very good way of comparing the libraries and the neural network, it can provide us with an insight into the libraries' performances. We used Google Colab which provides us with 12.7 GB of memory to run our code and record the runtime for calculation of embeddings.

Comparisons

We show comparison values only for the *Swiss Roll Curve* dataset. We decided not to include the Word2Vec dataset as the results were unsatisfactory.

- **Trustworthiness**
  Table 1 shows the different values of trustworthiness scores obtained by different libraries and approaches. The empty cells for 1M points show that the libraries were unable to run on the big datasets. Generally, an improvement from using 10k points can be recognized when using 100k points instead. The best trustworthiness scores for 100k and 1M points are highlighted in their corresponding columns.
- **Timings**
  Table 2 shows the timings taken by different libraries to calculate the embeddings. We do not include

Library Name	10k Points	100k Points	1M Points
Sklearn	0.5001	0.9916	-
Datafold	0.9993	0.9992	-
Datafold (Roseland)	0.9993	**0.9993**	0.9989
UMAP	**0.9998**	0.9801	-
Autoencoder	0.9994	0.9986	**0.9991**

Table 1: Comparison of Libraries on Different Datasets

UMAP here, as it uses GPU and would not be a fair comparison. Again, the empty cells for 1M represent that the library was not able to calculate the embeddings of the dataset. While Sklearn is by far the fastest library for 100k points, Datafold (Roseland) ranks second with less than one minute. Considering it also provides the best trustworthiness score, it can be seen as a winner on 100k points. For 1M points, however, Autoencoder performs better for both trustworthiness and computation time.
Note: The timings for Autoencoder is the training time for 50 epochs.

Library Name	10k Points	100k Points	1M Points
Sklearn	**1.3**	**10.0**	-
Datafold	6.2	485.1	-
Datafold (Roseland)	6.9	51.3	4357.8
Autoencoder	68.5	142	**1282.7**

Table 2: Time comparison of Libraries on Different Datasets(in seconds)

Report on task 5, Conclusion and future work

We started with the review of the megaman paper, but since it was not possible to install the library anymore, we explored various other available options in an effort to find an alternative. We tried different libraries and algorithms, focusing on their efficacy, computational efficiency, and scalability across two different datasets. Although we were unable to get good results on the Word2Vec dataset, we got pretty good results on the synthetic Swiss Roll dataset.

The Datafold library emerged as one of the most robust contenders with its diffusion maps algorithm. Its simple implementation and various utilities saved us a ton of time in tuning the parameters. While the original implementation of Diffusion Maps showcased commendable performance, it still has its limitations in handling very large datasets. Fortunately, datafold also had an alternative solution - the Roseland Model. This alternative significantly accelerated the embedding generation process while being computationally less resource intensive which allows us to run the model on not so powerful computers.

We were also able to show the efficacy of Autoencoders in dimensionality reduction. They provided us with a faster implementation with a comparable trustworthiness metric. While the libraries like sklearn and UMAP failed on 1M points, the autoencoders were able to learn latent representations.

In conclusion, our exploration illustrated the diverse landscape of dimensionality reduction techniques, showcasing the strengths and limitations of various libraries and algorithms.

There are still several options to improve these findings in future research. The first step is to get to know the dataset better and experiment with additional preprocessing. For the autoencoder, the model parameters can be fine-tuned using grid search or random search to achieve better results. Datafold struggled with a trade-off between speed and accuracy. In this aspect, a certain threshold could be determined depending on what the user deems more important.

References

[1] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006. Special Issue: Diffusion Maps and Wavelets.

[2] Datafol. Datafold. `https://datafold-dev.gitlab.io/datafold/index.htm`. Accessed on 06.02.2024.

[3] Almeida et al. The Eighteenth Data Release of the Sloan Digital Sky Surveys: Targeting and First Spectra from SDSS-V. *apjs*, 267(2):44, August 2023.

[4] geeksforgeeks. Swiss roll reduction with lle in scikit learn. `https://www.geeksforgeeks.org/swiss-roll-reduction-with-lle-in-scikit-learn/`. Accessed on 29.01.2024.

[5] Daniel Lehmberg, Felix Dietrich, Gerta Köster, and Hans-Joachim Bungartz. datafold: data-driven models for point clouds and time series on manifolds. *Journal of Open Source Software*, 5(51):2283, 2020.

[6] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[7] J. McQueen, M. Meilă, J. Vanderplas, and Z. Zhang. Megaman: Scalable manifold learning in python. *Journal of Machine Learning Research*, 17:1–5, 08 2016.

[8] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[9] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[10] Dominique Perraul-Joncas and Marina Meila. Non-linear dimensionality reduction: Riemannian metric estimation and the problem of geometric discovery, 2013.

[11] Datafol Roseland. Datafold roseland tutorial: S-curve digits. `https://datafold-dev.gitlab.io/datafold/tutorial_05_roseland_scurve_digits.html`. Accessed on 06.02.2024.

[12] scikit learn. sklearn.manifold.trustworthiness. `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.trustworthiness.html`. Accessed on 07.02.2024.

[13] Sloan Digital Sky Survey. Galaxy spectra. `https://skyserver.sdss.org/dr1/en/proj/advanced/galaxies/spectra.asp`. Accessed on 29.01.2024.