EXPLICIT NON-LINEAR DIMENSIONALITY REDUCTION

PIERRE VISCONTI

ABSTRACT. This paper tackles the challenge of dimensionality in machine learning through the use of an explicit manifold learning algorithm with a primary objective in reducing both computational time and storage requirements. By incorporating a polynomial mapping assumption, the algorithm simplifies complex datasets before their utilization in machine learning processes. This approach not only enhances the representation of intricate data structures compared to traditional linear mapping methods but also ensures the practical applicability of the algorithm through the production of an explicit model. The computational time and storage requirements with the reduced dataset produced by the manifold learning algorithm are compared on various machine learning algorithms for both training data and labeling of out of sample data.

1. INTRODUCTION

1.1. **Motivation.** Machine learning data is often complex and of high dimension. Consider an example dataset comprised of grey-scale images of peoples faces, where each image is 200x300 pixels in size, as shown in *Figure 1.* We typically want data to be represented as a matrix where each row is a single data point (often referred to as an observation), and each column represents a feature or variable. To format the dataset of images in this manner, each image is thought of as a matrix and is flattened into one long vector to be represented as a single row. For our example dataset, since we have 200x300 pixel images giving a total of 60,000 pixels, we have 60,000 features in our dataset, which we refer to as the number of dimensions. The high dimensionality of the data poses problems by the high computational cost in analyzing the data and training a model, often referred to as the curse of dimensionality.

FIGURE 1. Example dataset comprised of grey-scale images assumed to be 200x300 pixels. Adapted from [3].

To illustrate the problem, consider the time complexity of some popular machine learning algorithms, given in big-O notation, which returns the number of operations required to train the algorithm as a function of the number of observations (n) and the number of features (m) of the dataset.

$$\text{Decision Trees: } O(mn \log(n))$$

$$\text{Naive Bayes: } O(mn)$$

As the dimensionality of the dataset increases the number of operations increases proportionally, as a best case scenario. With our example dataset of 60,000 dimensions, it is clear that the number of operations required to train such machine learning algorithms with it would be high, and it would be desirable to lower the number of dimensions. Suppose that for our specific application with the dataset we are only interested in two variables, the emotion on the person's face, and the rotation of their face. This can be represented as a point in two-dimensional space, as seen in *Figure 2*. Meaning the dataset has been reduced from 60,000 dimensions to two, thereby significantly reducing the number of operations required to train the machine learning algorithm. This concept is known as dimensionality reduction.

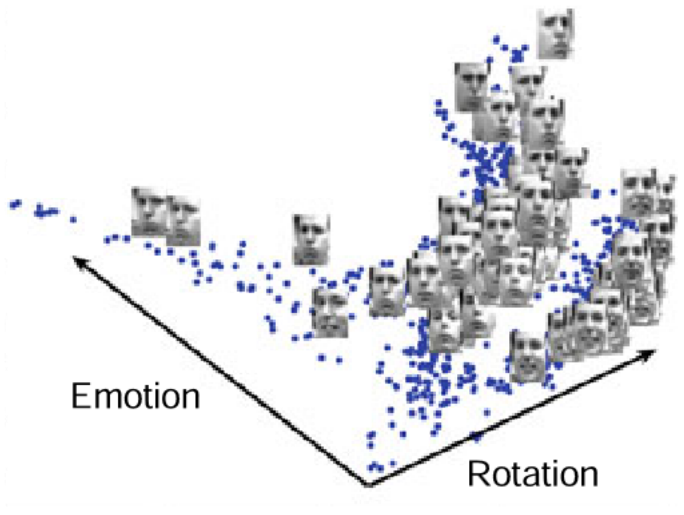


FIGURE 2. Example dataset comprised of grey-scale images represented in $\mathbf{R}^2$. Adapted from [8].

1.2. **Background.** Dimensionality reduction is the idea that when the data is of artificially high dimension, it can be reduced to a lower dimension while still retaining the intrinsic properties of the data. Dimensionality reduction algorithms aim to construct a mapping from the high dimensional data to its corresponding low dimensional representation [5]. Linear techniques such as Principal Components Analysis (PCA) have been well known for some time, but struggle with complex non-linear data [11]. Non-linear methods, known as Manifold learning, can handle complex data, however most existing algorithms do not build an explicit model, meaning new incoming data samples must be added to the original data and the dimensionality reduction algorithm run again on the entire dataset, thereby defeating the intended purpose of lowering the overall computational complexity. Additionally, many manifold learning algorithms assume a linear mapping which may not perform as desired [8] [7]. To best reduce the complexity of the data in a way that is practical for machine learning, we analyze an explicit manifold learning algorithm that assumes a polynomial mapping rather than a linear mapping. The following section defines what we mean by a mapping and provides a more formal definition of manifold learning.

2. Mathematical Background

2.1. **Manifold Learning.** Manifold learning algorithms all rely on a basic assumption known as the Manifold Assumption. The Manifold Assumption states that the data must be sampled from a distribution that is close to or supported on, a smooth manifold (later defined in 2.2) of dimension d, embedded in $\mathbf{R}^n$. The dimension d of a manifold is called the intrinsic dimension, which refers to the minimum number of dimensions required to specify a point on it. The manifold learning algorithm aims to find a mapping F (formally called an embedding) of a high dimensional point in $\mathbf{R}^n$, to a lower dimensional point in $\mathbf{R}^m$ [8], which we informally denote as $\mathbf{R}^n \to \mathbf{R}^m$. Notably, we assume that the high dimension n is greater than the intrinsic dimension of the manifold, d, otherwise it would not make sense to further reduce the dimensionality of the data in the context of machine learning. It is also desirable that the dimension the data is mapped to, m, is greater or equal to d, otherwise the reduced dimension is lower than the intrinsic dimension. Logically, n must also be greater than m. This is summarized by lemma 2.1.

Lemma 2.1. *The following properties of dimensions $n, m,$ and d must be observed in constructing a mapping F.*

- *Assert that $n > m$.*
- *Assume that $n > d$*
- *Desire that $m \geq d$.*

2.2. **Manifolds.** We define what is meant by a smooth manifold in definition 2.2. For the application of this paper, a smooth manifold can be summarized as a topological space that has a differentiable structure and is locally Euclidean, meaning it resembles Euclidean space near every point.

Definition 2.2. M is a smooth manifold of dimension d if it is a topological space where,

- For every $p, q \in M$ there exists disjoint open subsets $U, V \subseteq M$, such that $p \in U$ and $q \in V$.
- There exists a countable basis for the topology of M.
- For every $p \in M$, there exists
  - an open subset $U \subseteq M$ containing p,
  - an open subset $\hat{U} \subseteq \mathbf{R}^d$, and
  - a homeomorphism $\phi : U \to \hat{U}$.

Adapted from [4].

2.3. **Embeddings.** A smooth map, as defined in 2.3, can be thought of as a function that is infinitely differentiable [2]. In the context of smooth manifolds, we define an embedding as a smooth map between two manifolds whose inverse exists and is also smooth [5], which maps elements from one space to another. This is defined more formally in 2.4.

Definition 2.3. A map $F : N \to M$ between manifolds is smooth at $p \in N$ if there are coordinate charts, defined as a mapping that takes an open subset of a manifold and provides a one-to-one correspondence with an open subset of Euclidean space, (U, ϕ) around p and (V, ψ) around $F(p)$ such that $F(U) \subseteq V$ and such that the following composition is also smooth

$$\psi \circ F \circ \phi^{-1} : \phi(U) \to \psi(V).$$

The function F is a smooth map from N to M if it smooth for all $p \in N$. Adapted from [2].

Definition 2.4. A smooth map $F : N \to M$ between smooth manifolds is an embedding if it satisfies the following two properties.

- Letting I be the image of F , the map F is a homeomorphism between N and I.
- Let $G : I \to N$ be the inverse of F . Then G is smooth.

Adapted from [6].

Importantly, embeddings have several properties that we are interested in. Embeddings are injective and they preserve the geometric structure of the object being embedded. In the context of manifold learning this means that the data points close to each other in the high dimensional space remain close in the low dimensional space. Naturally, points far away from each other should remain far away as well [8]. Thus, the algorithm should aim to find an mapping where this holds true, in order to find the embedding. Notably, we also desire that the embedding is found explicitly and is non-linear in nature, to obtain better results in a way that is practical for machine learning.

3. Developing an Algorithm

3.1. **Algorithm Goals.** Suppose that we begin with a dataset X that is of high dimension. Before feeding it to the machine learning algorithm, the manifold learning algorithm is used on X to reduce the dimensionality, producing a new set of data Y. This process is visualized in *Figure 3*. The machine learning algorithm is trained on Y to produce a model that takes some observation y_i as an input and assigns it a label. Suppose that x_i is a new observation that was not originally included in X, and thus was never reduced in dimension. The model produced by the machine learning algorithm can only label the reduced observation y_i, therefore x_i must be reduced in order to be assigned a label. If the manifold learning algorithm does not build an explicit model, x_i must be added to X and the algorithm run again over all of X just to reduce our new observation x_i. The term explicit model refers to having a closed form expression that is independent of the training samples. This is required for the manifold learning algorithm to be useful as a realistic application to machine learning.

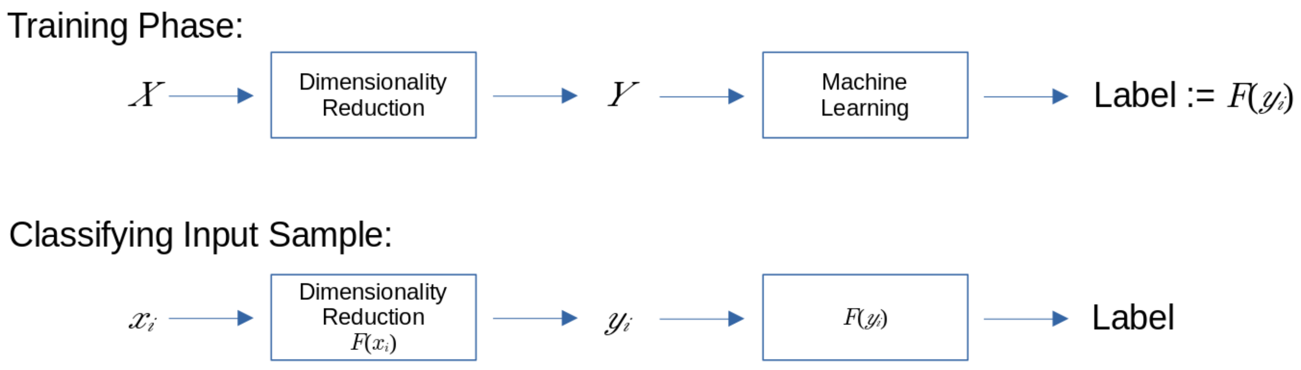


Figure 3. Flow chart of dimensionality reduction as a pre-processing step for machine learning.

Some linear methods such as NPP and NPE do indeed build explicit models. They define the linear mapping from a high dimensional point x_i to its low dimensional representation y_i as

$$y_i = U^T x_i, \text{ where } U \in \mathbf{R}^{n\times m}.$$

While these methods do produce an explicit model, due to the complex nature of data in machine learning, a linear mapping is likely not sufficient to produce desirable results, see [8] and [7] for more details. Other algorithms do exist though. A popular algorithm, Locally linear embedding (LLE), does not necessarily assume a global linear mapping, but does assume a local linear mapping and no explicit model is produced. In its final step, LLE builds an embedding by solving an optimization problem where linear weights are fixed while the low dimensional coordinates y_i are optimized to minimize error. The nature of the embedding is completely unknown and additionally the algorithm builds the weights by assuming that each point x_i in the high dimensional space can be reconstructed by a linear combination of its neighbors which may not be the case [8].

The importance of a non-linear global embedding is demonstrated in *Figure* 4 where the reduced data from four manifold learning algorithms are compared on a toy dataset comprised of points in three dimensional space; note that X is the training set and Z the expected output. The results produced by the linear algorithms (NPP, ONPP), denoted as Y_{NPP} and Y_{ONPP} do not manage to unfold the underlying object in the training set X, while the non-linear manifold learning algorithms LLE and NPPE do. Importantly we note that NPPE, which assumes a global polynomial mapping, significantly out performs LLE on this dataset, which has an unknown global embedding but local linear relationships [8]. Therefore, we analyze a manifold learning algorithm that assumes an explicit non-linear mapping instead.

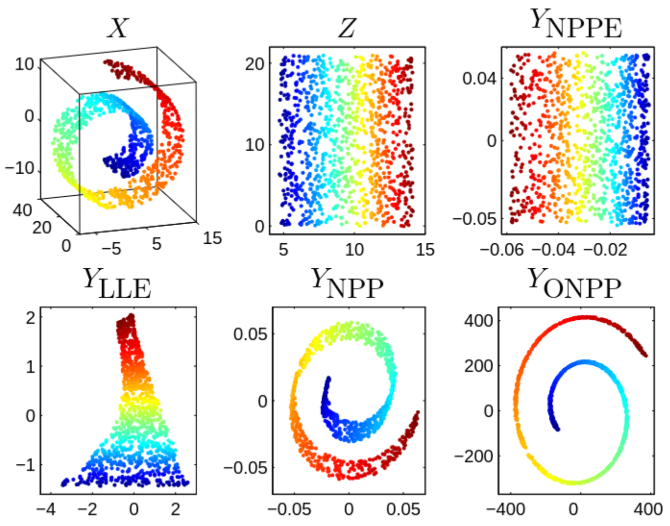


FIGURE 4. Comparing linear and non-linear algorithms on unfolding surfaces embedded in $\mathbf{R}^3$. Adapted from [8].

3.2. **LLE.** While we desire a polynomial mapping assumption, we first work through the steps of LLE in greater detail. Later we define what is meant by a polynomial mapping assumption and how it is applied to LLE to create a new algorithm as presented by the authors in [8] and [7].

Algorithm 1 Locally Linear Embedding

```
Input: Data matrix $X$ of size $n \times N$, the number $k$ of nearest neighbors, and the desired number of dimensions $m$ to map to.
Output: Data matrix $Y$ of size $m \times N$.
1. Compute the k-nearest neighbors of each data point $x_i$.
2. Compute the weights $R_{ij}$ that best reconstructs each data point $x_i$ by its neighbors to form the linear weight matrix $R$.
3. Compute the vectors $y_i$ best reconstructed by the weights $R_{ij}$.
```

Adapted from [9] and [10].

We begin by assuming we have some data set $\mathcal{X} := \{x_1, x_2, ..., x_N\}$ in the high dimensional space $\mathbf{R}^n$, which we aim to represent as $\mathcal{Y} := \{y_1, y_2, ..., y_N\}$ in $\mathbf{R}^m$ where $n > m$. The datasets are represented as matrix data structures with sizes $n \times N$ and $m \times N$, respectively. The dimension m that we wish to map to is passed as an argument to the algorithm which the user can choose the value of. Using the property that embeddings preserve geometric structure (described in section 2.3), LLE aims to preserve local relationships between data points [8]. It does this by first finding a set of linear coefficients that best reconstructs each data point x_i by its k-nearest neighbors. We note that the number k of nearest neighbors is also passed as an argument and can be tuned to provide better results. Using Euclidean Distance, the linear reconstruction weights $R_{ij}, i, j = 1, 2, ..., N$ are given by minimizing the sum of the squared distances between all the data points and their reconstructions, subject to the constraints that x_i is only reconstructed from its neighbors, and the weights for x_i must sum to one. This is given by the following optimization problem [9] [8] [10].

$$\text{(i)} \qquad R_{ij} = \underset{\sum_{j=1}^{N} R_{ij}=1}{\arg\min} \quad \sum_{i=1}^{N} \left\| x_i - \sum_{j=1}^{N} R_{ij} x_j \right\|_2^2$$

Once the linear weight matrix R has been constructed, LLE constructs a neighborhood preserving mapping (an embedding) by fixing the weights R_{ij}, while optimizing the coordinates y_i to minimize error, subject to several constraints [8]. The first constraint is called unit-covariance, which serves to normalize the data in the embedding space, meaning the reconstruction errors are measured on the same scale for different coordinates. The second constraint is referred to as zero mean, which is used to center the coordinates on the origin [9]. This is given by the following optimization problem [9] [8] [10].

$$\text{(ii)} \qquad \min \quad \sum_{i=1}^{N} \left\| y_i - \sum_{j=1}^{N} R_{ij} y_j \right\|_2^2$$

$$\text{s.t.} \quad \frac{1}{N} \sum_{i=1}^{N} y_i y_i^T = I_m$$

$$\sum_{i=0}^{N} y_i = 0$$

At this point the reduced dataset $\mathcal{Y}$ has been computed and the algorithm terminates.

3.3. **Polynomial Mapping Assumption.** To produce a manifold learning algorithm that assumes an explicit non-linear mapping, the authors of [8] and [7] modify the presented algorithm above. Specifically, they define a non-linear mapping as the following. Given a data set $\mathcal{X} := \{x_1, x_2, ..., x_N\}$ in the high dimensional space $\mathbf{R}^n$, assume there exists an explicit polynomial mapping from $\mathcal{X}$ to its low dimensional representation $\mathcal{Y} := \{y_1, y_2, ..., y_N\}$ in $\mathbf{R}^m$. The k-th component of $y_i \in \mathcal{Y}$ is defined as a polynomial of degree p with respect to x_i, such that

$$y_i^{(k)} = v_k^T \phi(x_i),$$

where $v_k \in \mathbf{R}^{pn}$ and $\phi(x)$ is given by definition 3.1.

Definition 3.1. For a given data vector $x_i \in \mathcal{X}$, the mapping $\phi : \mathbf{R}^n \to \mathbf{R}^{pn}$ is defined as

$$\phi(x_i) = \begin{bmatrix} \overbrace{x_i \odot x_i \odot \cdots \odot x_i}^{p \text{ times}} \\ \vdots \\ x_i \odot x_i \\ x_i \end{bmatrix}.$$

Definition 3.2. The Hadamard product of two $m \times n$ matrices A and B is defined as

$$A \odot B = \begin{bmatrix} a_{11}b_{11} \cdots a_{1n}b_{1n} \\ \vdots \\ a_{m1}b_{m1} \cdots a_{mn}b_{mn} \end{bmatrix}.$$

We note that the $\odot$ operator in definition 3.1 represents the Hadamard product, given by definition 4.1. In this case, since x_i is a column vector, $\phi(x_i)$ becomes one long column vector where if x_i has size n, then the first n components of $\phi(x_i)$ are x_i raised to the p^{th} degree. The next n components of $\phi(x_i)$ would then be x_i raised to the $(p-1)^{\text{th}}$ degree, and so on. Consider the following example: let $x_1 = [4 \quad 2 \quad 3]^T$ and $p = 3$. Then $\phi(x_1)$ is

$$\phi(x_1) = \left[4^3 \quad 2^3 \quad 3^3 \quad 4^2 \quad 2^2 \quad 3^2 \quad 4 \quad 2 \quad 3\right]^T = [64 \quad 8 \quad 27 \quad 16 \quad 4 \quad 9 \quad 4 \quad 2 \quad 3]^T .$$

The polynomial mapping assumption is then applied to LLE which produces the Neighborhood Preserving Polynomial Embedding (NPPE) algorithm.

4. NPPE Algorithm

4.1. **LLE Modifications.** The polynomial mapping assumption in section 3.3 is first applied to the process of computing the linear reconstruction weights R_{ij} given by (i). We now desire to find the weights under the polynomial mapping assumption, which we will denote as the non-linear weights W_{ij}. Algebraically, it is shown in [8] that the values of the non-linear weights depend on the linear weights, thus the first step of NPPE is to compute the linear weight matrix R just like in LLE. Rather than solving the optimization problem (i), the linear weight matrix $R = [r_1 \quad r_2 \ \cdots \ r_N]$ in $\mathbf{R}^{N\times N}$, is given by computing the following closed formed solution for r_i

$$r_i = \frac{G^{-1}e}{e^T G^{-1} e}, \text{ such that} \tag{1}$$

- e is a column vector of all ones,
- $G_{jl} = (x_j - x_i)^T(x_l - x_i)$ where x_j, x_l are in the k-nearest neighbors of x_i [8] [7] [10].

Once the linear weight matrix R has been constructed, the non-linear weights are then simply computed by the following equation [8] [7]

$$W_{ij} = R_{ij} + R_{ji} - \sum_{k=1}^{N} R_{ik}R_{kj}, \text{ and } \sum_{j=1}^{N} W_{ij} = 1. \tag{2}$$

The polynomial mapping assumption is then applied to the second step of LLE. The detailed mathematical steps in the process, mostly algebra and redefining of variables, are given in [8]. We define $\phi = [\phi(x_1) \quad \phi(x_2) \ \cdots \ \phi(x_N)]$ to be a $(pn) \times N$ matrix, where $\phi(x_i)$ is given by definition 3.1 and let W be the non-linear reconstruction weight matrix of size $N \times N$. The optimization problem (ii) turns into the following, where D is the identity matrix [8] [7]

$$\min_{v_k} \quad \sum_k v_k^T \phi(D-W)\phi^T v_k \tag{iii}$$

$$\text{s.t.} \quad v_j^T \phi D \phi^T v_k = \delta_{jk}.$$

Let A and B be matrices of size $(pn)\times(pn)$, where $A = \phi(D-W)\phi^T$ and $B = \phi D \phi^T$. Then the optimization problem above can be represented as the following

$$\min_{v_k} \quad \sum_k v_k^T A v_k$$

$$\text{s.t.} \quad v_j^T B v_k = \delta_{jk}.$$

We note that the constraint δ_{jk} is the Kronecker delta, which is 1 if (j = k) and 0 otherwise, importantly it is always equal to a constant. According to definition 4.1, the optimization problem above has the equivalent form given by the Generalized Rayleigh-Ritz Quotient. According to [1], this is equivalent to solving a generalized eigenvalue problem, as defined in definition 4.2.

Definition 4.1. Generalized Rayleigh-Ritz Quotient. The generalized Rayleigh-Ritz Quotient is defined, for symmetric matrices A and B, nonzero vector x, and constant c, as

$$R(A,B;x) := \frac{x^TAx}{x^TBx},$$

which has the following equivalent form

$$\min / \max_{x} \quad x^TAx$$

$$\text{s.t.} \quad x^TBx = c.$$

Adapted from [1].

Definition 4.2. Generalized Eigenvalue Problem. The generalized eigenvalue problem of two symmetric matrices $A \in \mathbf{R}^{d\times d}$ and $B \in \mathbf{R}^{d\times d}$ is defined as:

$$Av_i = \lambda_i Bv_i, \quad \forall i \in 1, ..., d$$

or in matrix form as

$$Av = \lambda Bv,$$

where $v = [v_1 \ \cdots \ v_d]$. Adapted from [1].

Therefore our optimization problem (iii) is equivalent to solving the following generalized eigenvalue problem [8] [7]

$$\phi(D-W)\phi^Tv_i = \lambda\phi D\phi^Tv_i, \quad v_i^T\phi D\phi^Tv_j = \delta_{ij}. \tag{3}$$

A generalized eigenvalue problem is related to the common eigenvalue problem $Av = \lambda v$, where an eigenvalue problem is a special case of the generalized version with matrix B equal to the identity matrix. The solutions to a generalized eigenvalue problem are given by several different algorithms, depending on certain properties of matrices A and B, see [1] for more detailed information. For our implementation of the manifold learning algorithm, discussed later in the text, the eigenpairs were found using a solver in a C++ library. In this situation, since we are minimizing the objective function, the optimal solutions are given by the eigenvectors $v_i, i = 1, 2, ..., m$ corresponding to the m smallest eigenvalues [8] [7].

For a given data point $x_i \in \mathcal{X}$, its low dimensional representation $y_i \in \mathcal{Y}$ is then given by

$$y_i = \begin{bmatrix} v_1^T\phi(x_i) & v_2^T\phi(x_i) & \cdots & v_m^T\phi(x_i) \end{bmatrix}^T, \tag{4}$$

where v_i for $i = 1, 2, ..., m$ are the eigenvectors computed from (3). Importantly, the mapping function above holds true for a new data sample $x_t \notin \mathcal{X}$, assuming the new data sample lives on or near our manifold, allowing for its low dimensional representation y_t to be computed efficiently without having to run the algorithm again [8] [7]. This is our desired explicit non-linear embedding.

4.2. **Algorithm Summary.** The Neighborhood Preserving Polynomial Embedding (NPPE) algorithm, developed by the authors in [8] [7], is summarized by algorithm 2 on the next page. We note that the numbers corresponding to the steps in the summary, align with the steps, (1), (2), (3), and (4), covered in the previous section, 4.1.

5. Results

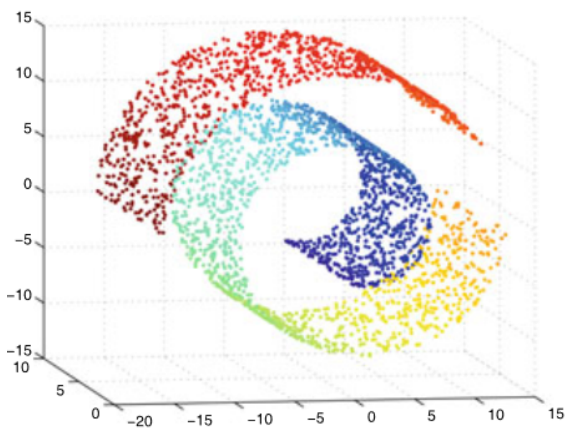
The results are split into three subsections. First we discuss the results of the NPPE algorithm. Then we discuss our own implementation and our results. Later we analyze the use of the algorithm as a pre-processing step for a machine learning algorithm.

Algorithm 2 Neighborhood Preserving Polynomial Embedding

```
**Input:** Data matrix $X$ of size $n \times N$, the number $k$ of nearest neighbors, the desired number of dimensions $m$ to map to, and the polynomial degree $p$.
**Output:** Data matrix $Y$ of size $m \times N$ and an explicit mapping function to determine $y_i$ given $x_i$.
1. Compute the linear weight matrix $R$.
2. Compute the non-linear weight matrix $W$.
3. Solve the generalized eigenvalue problem to get the eigenvectors $v_i, i = 1, 2, ..., m$.
4. Map the high-dimensional data to the low-dimensional embedding space using the produced mapping function.
```

Adapted from [8] and [7].

5.1. **NPPE Results.** We begin by providing the results that the authors in [8] and [7] achieve with their implementation of the NPPE algorithm, discussed in sections 4.1 and 4.2. The algorithm is applied on a toy dataset, shown in *Figure* 5a, comprised of points in three dimensional, to find an explicit non-linear embedding from $\mathbf{R}^3 \rightarrow \mathbf{R}^2$. The data is split into a training set, the lower part, and a testing set, the upper part. The parameter of k-nearest neighbors is set to be 1% of the training samples N, the polynomial degree p to two, and m to two [7]. The embedding is used to reduce the original dataset, producing the results in *Figure* 5b. We note that the algorithm is not only able to unfold the data well for the training set, but importantly for the testing set as well. It also manages to capture the gap between the training and testing sets which demonstrates the efficacy of the algorithm.


(A) SwissRoll dataset in $\mathbf{R}^3$

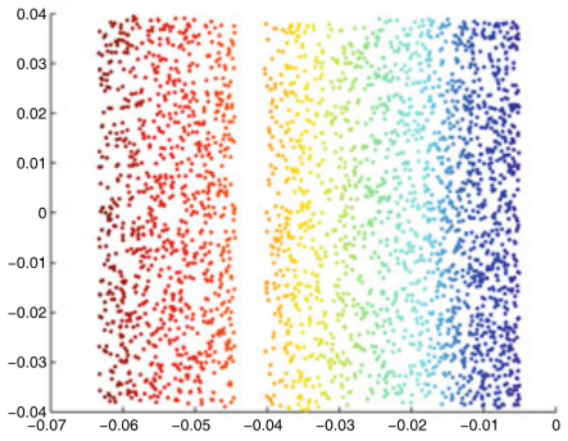
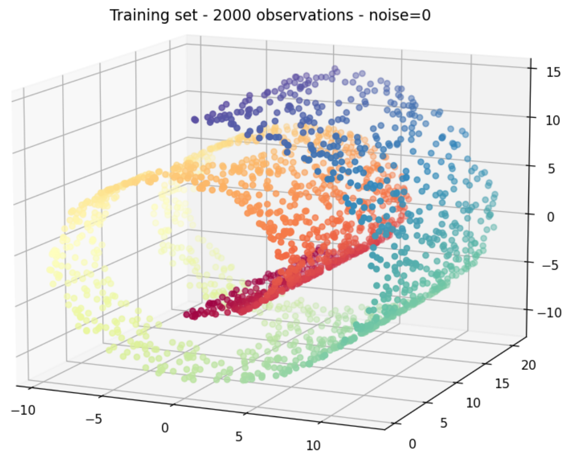

(B) SwissRoll representation in $\mathbf{R}^2$

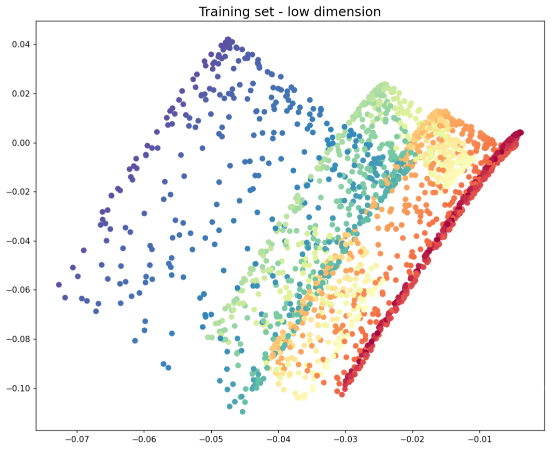
FIGURE 5. Performance of the NPPE algorithm on the SwissRoll dataset. Adapted from [7].

5.2. **NPPE Implementation.** The NPPE algorithm discussed in sections 4.1 and 4.2 was then implemented from scratch in C++ using Eigen3 for matrix operations and OpenMP for parallel programming. The code for the project can be found on Github at https://www.github.com/petrosvisconte/nppe. Since C++ was used as the programming language and the algorithm had to be built from scratch, coding optimizations were performed manually. A lot of attention was spent in parallelizing the computations to reduce the computation time, which proves complicated when many functions and calculations access and modify the same locations in memory. Additionally, parts of the code were written in a manner for the compiler to best auto vectorize the loops, making use of the AVX instruction sets on the CPU. Memory usage was also monitored; solving the generalized eigenvalue problem, using a solver from Eigen3, is by far the most memory intensive portion of the code. For the following results, the algorithm was run on an AMD 5950x CPU with 16 physical cores (32 logical cores), equipped with 32GB of DDR4 memory. 128GB of memory was required for datasets reaching 1GB or more in size. We note that the overall load average on the processor was near 100%, indicating the algorithm makes close to full use of the CPU cores.

Our implementation was applied to a SwissRoll dataset with a hole added in the middle to increase difficulty, as shown in *Figure* 6a, to produce the reduced dataset shown in *Figure* 6b. The number k of nearest neighbors was set to 1% of the observations N and the polynomial degree p to two. Unfortunately, our output does not exactly match the expected output shown in *Figure* 5a, due to a bug in the code. Nevertheless, it can be seen that our implementation is still behaving as expected in several aspects, and notably it captured the added hole. We then produced a new dataset with added noise and filled in the hole, shown in *Figure* 7a. The goal from doing so was to see if the explicit embedding produced could still perform as expected on data points that no longer lie exactly on the manifold used for training. The output is shown in *Figure* 7b, and demonstrates that the embedding is robust in that regard.

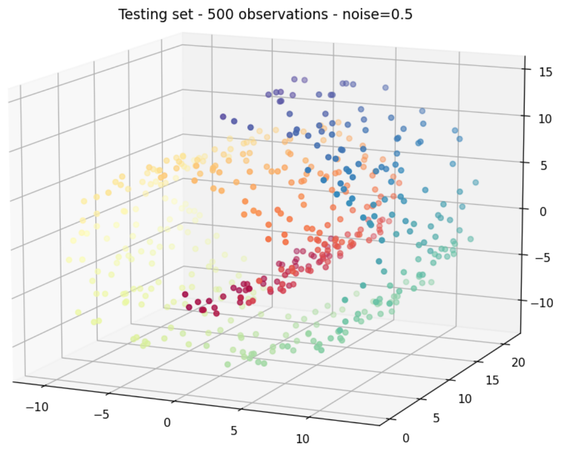


(A)

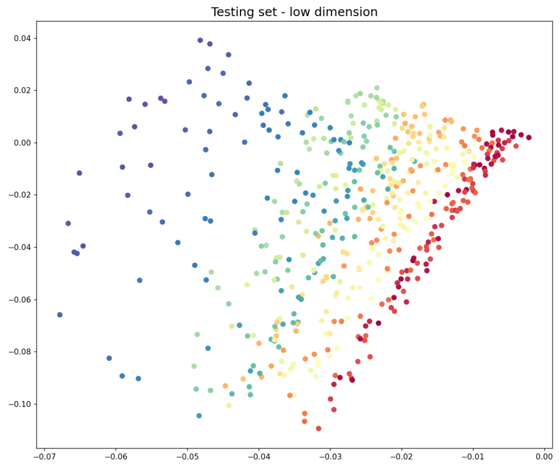


(B)

Figure 6. Performance of our implementation on the SwissRoll training set with a hole.



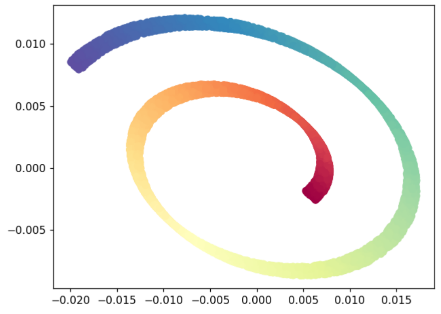
(A)



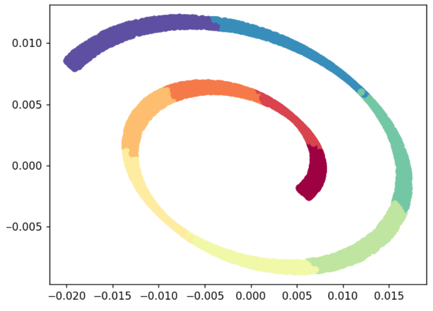
(B)

Figure 7. Performance of our implementation on the SwissRoll testing set with added noise and no hole.

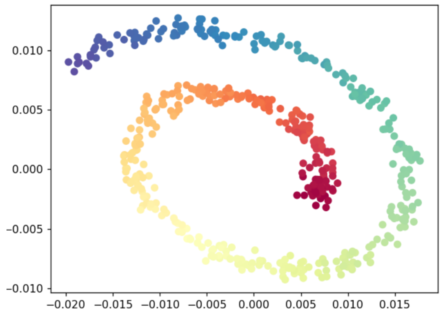
5.3. **Machine Learning.** We began with a regular SwissRoll dataset comprised of 10,000 observations to use as the training set, and a SwissRoll dataset with a hole and added noise, comprised of 500 observations, to serve as the testing set. We then applied our implementation of NPPE to the training set to find an embedding from $\mathbf{R}^3 \to \mathbf{R}^2$. Both the training and testing sets were then reduced in dimensionality using the embedding produced from the training set. This was initially performed with $p = 2$ and then repeated with $p = 1$. It should be noted that with $p = 1$, the embedding is linear instead of polynomial, so the results should be similar to linear algorithms such as NPP or ONPP. The results for $p = 1$ are shown in *Figure* 8a and *Figure* 8c (the algorithm performs as expected with $p = 1$).



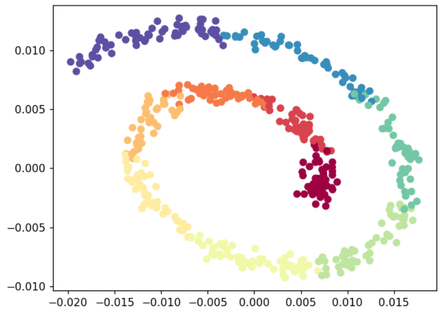
(A) Reduced SwissRoll in $\mathbf{R}^2$ - Training.



(B) Binned training set.



(C) Reduced SwissRoll in $\mathbf{R}^2$ - Testing.



(D) Binned testing set.

FIGURE 8. Performance of our implementation on the SwissRoll testing set with added noise and no hole.

Both reduced datasets were then binned by color in order to assign a discrete color label to all of the points, as shown in *Figure* 8b and *Figure* 8d. The reduced training set was then fed to a deep neural network, using tensorflow and sklearn in Python, to predict the color of the point based on the coordinates of the point in $\mathbf{R}^2$. The model produced by the neural network was then used to assign a color to the reduced testing set and compared against their actual colors to determine the accuracy. This was compared against the neural network's performance when training on the non-reduced SwissRoll dataset with points in $\mathbf{R}^3$. The results are summarized in *Table* 1

Test, set	Accuracy (%)
Baseline, train	96.6
Baseline, test	96.4
$p = 2$, train	81.4
$p = 2$, test	59.9
$p = 1$, train	95.6
$p = 1$, test	95.2

TABLE 1. Accuracy of the Neural Network predictions.

6. Conclusions

This paper has presented a comprehensive exploration of the Neighborhood Preserving Polynomial Embedding (NPPE) algorithm, its practical implementation, and its application as a preprocessing step for machine learning tasks. The NPPE algorithm has demonstrated its effectiveness in unfolding complex datasets, such as the SwissRoll dataset, from higher to lower dimensional spaces while preserving neighborhood structures. The algorithm's ability to maintain the integrity of the data structure during the dimensionality reduction process, even when faced with added noise and alterations in the dataset, underscores its potential for real-world applications in the realm of machine learning.

While the algorithm itself and the embedding produced appears to be robust, the results using our C++ implementation were not good enough to properly test NPPE as a pre-processing step on complex image based datasets with many dimensions. Due to a bug with our implementation, we were reduced to dealing with datasets of three dimensions at most, to verify the output of the manifold learning step. Even so, the results produced on the SwissRoll with a polynomial degree of two were not good enough to maintain an accuracy in the machine learning model comparable to the baseline results. We noticed that with a polynomial degree of one, our implementation would produce better results (due to the bug being present when $p > 1$). Thus, reverting to a linear embedding restored the performance to near-baseline levels which demonstrates that reduced datasets can still provide the necessary data for the machine learning algorithm to effectively learn on. We expected to notice a decrease in computation time with the reduced dataset but found the training time of the machine learning step to be within the margin of error using both methods. This is likely due to the Python libraries used being extremely well optimized and that we only reduced our dataset by a single dimension, which was not enough to sufficiently reduce the overall number of operations required. We still expect there to be a drop in computation time for image based datasets with many dimensions being reduced in complexity.

Future work could focus on refining the implementation to address the discrepancies observed in the output, optimizing the algorithm for larger datasets, and exploring the impact of different polynomial degrees on various types of data. Additionally, integrating the NPPE algorithm into a broader range of machine learning workflows could further validate its utility and scalability. Overall, we hope the findings from this study motivate further development and research of explicit non-linear dimensionality reduction algorithms and their applicability to machine learning.

References

[1] Benyamin Ghojogh, Fakhri Karray, and Mark Crowley. Eigenvalue and generalized eigenvalue problems: Tutorial. 2019.

[2] Marco Gualtieri. Differential geometry course notes, Accessed 2024. University of Toronto Department of Mathematics.

[3] Abdullah Ghanim Jaber, Ravie Chandren Muniyandi, Opeyemi Lateef Usman, and Harprith Kaur Rajinder Singh. A hybrid method of enhancing accuracy of facial recognition system using gabor filter and stacked sparse autoencoders deep neural network. *Applied Sciences*, 12(21), 2022.

[4] J.M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2003.

[5] Marina Meilă and Hanyu Zhang. Manifold learning: What, how, and why. *Annual Review of Statistics and Its Application*, 11(Volume 11, 2024):393–417, 2024.

[6] Andrew Putman. A geometrically-minded introduction to smooth manifolds. Lecture Notes, University of Notre Dame, 2016.

[7] Hong Qiao, Chao Ma, and Rui Li. *The "Hand-eye-brain" System of Intelligent Robot: From Interdisciplinary Perspective of Information Science and Neuroscience*, chapter Explicit Nonlinear Mapping for Manifold Learning with Neighborhood Preserving Polynomial Embedding, pages 81–106. Springer Singapore, 2022.

[8] Hong Qiao, Peng Zhang, Di Wang, and Bo Zhang. An Explicit Nonlinear Mapping for Manifold Learning. *IEEE Transactions on Cybernetics*, 43(1):51–63, 2013.

[9] Lawrence Saul and Sam Roweis. An Introduction to Locally Linear Embedding. *Journal of Machine Learning Research*, 7, 2001.

[10] Cosma Rohilla Shalizi. Lecture 14: Data mining. Online, 2009. Accessed: 2024-05-02.

[11] Laurens van der Maaten, Eric Postma, and H. Herik. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research - JMLR*, 10, 01 2007.