Unsupervised Learning Project

Erika Lena

Spring 2023

Abstract

The following paper contains a discussion of the method proposed by Michael Gashler in his paper: **Iterative Non-linear Dimensionality Reduction by Manifold Sculpting** [1]. The method is exposed and compared with other manifold learning techniques seen during lectures; an implementation along with the results obtained is shown.

1 Introduction

Manifold sculpting is an iterative non-linear dimensionality reduction technique for manifold learning, proposed in 2007 by Michael Gashler.
It falls into the category of NLDR (Non-Linear Dimensionality Reduction), where the non-linearity is given by the assumption that the relationships between neighboring points are more important than those between distant points, and they need to be preserved and handled with higher attention.
In general dimensionality reduction techniques consist of two steps: the transformation of our data and their projection in fewer dimensions.
The main point is of course the first one, in which we decide how to transform our data before projecting them.
In what follows, it is explained what **manifold sculpting** does to transform the data and the method is compared with others NLDR techniques.

2 How it works

The key idea behind manifold sculpting is to preserve **local structure** of our dataset, where by local structure it considers not only neighborhood distance, but also curvatures.
In other words, manifold sculpting finds for each point its k nearest neighbors and then it proceeds by computing a set of relationships between these neighbors.
In particular, for each pair (p_i, n_{ij}), made of a point p_i in our dataset and its $j-th$ neighbor (between the k nearest ones), it searches for m_{ij} which is the most colinear point to p_i through n_{ij}.
The preprocessing of the data consists of what just explained and the optional alignment of the axes with principle components, which can be found, for example, through PCA.
Then the algorithm proceeds by scaling out data in the dimensions to be discarded, while increasing their size in the ones to be preserved, until a certain stopping criteria is met,
In this way, data can be projected in the end, without loosing too much information; however, by doing so, local relationships are lost, and so

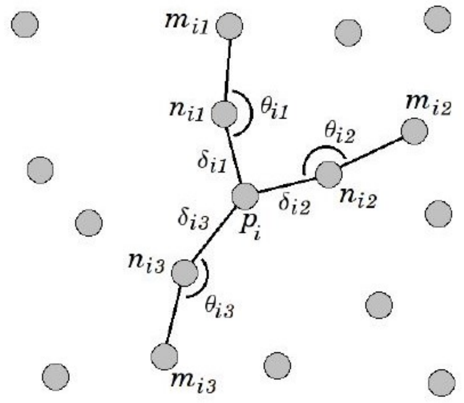


Figure 1: δ and θ define the relationships that Manifold Sculpting attempts to preserve.

at each iterative step the points are adjusted in
order to restore them.
The points are adjusted in order to minimize the following cost function:

$$\epsilon_{p_i} = \left(\left(\frac{\delta_{ij} - \delta_{ij_0}}{2\delta_{avg}}\right)^2 + \left(\frac{\theta ij - \theta_{ij_0}}{\pi}\right)^2 \right) \tag{1}$$

It expresses the error committed in representing original local structures after having moved the points. Each point p_i has its own unique error surface and this helps in avoiding local minima of the whole error function.

Manifold Sculpting Algorithm

1. Find the k nearest neighbor of each point.
2. Compute a set of relationships between neighbors.
3. Optionally align axes with principal components.
4. While the stopping criteria has not been met:
   (a) Scale the data in the non-preserved dimensions.
   (b) Adjust points to restore the relationships.
5. Project the data.

3 Implementation

The Manifold Sculpting algorithm has been implemented in Python (3.9) and the code is available here.
Conceptually, the algorithm is very simple and easy to be understood, however there are a lot of small details that can be (or need to be) properly tuned.
The main class which implements the algorithm is available in */src_code/manifold_sculpt.py* and an IPython Notebook is provided to illustrate it step by step.
The algorithm is made essentially of the following functions, provided in the class **ManifoldSculpting**, along with the function which provides local relationships.

```
def get_local_relationships(data, k=10);

class ManifoldSculpting:
    def __init__(self, n_components, k=10, n_iter=100, sigma = 0.9,
    th = 10**(-3), align=False, verbose=False);

    def fit(self, data);

    def compute_error(self, data, curr_idx, adj_data);

    def adjust_points(self, data, curr_idx, eta, adj_data);
```

The first function implements steps 1 and 2, by finding local neighbors and colinear points.
In the class **ManifoldSculpting**, we have the core function which *fits* the data, by taking the dataset and returning its dimensional reduction. It calls function one to get the data structures needed to represent the local structure of the dataset. Then it optionally calls PCA to get align dimensions to principal components, and it starts the iterative procedure.

This one is made of the two steps already discussed, the first is to scale the data in order to reduce the amount of information along the dimensions to be discarded and the latter is to restore local relationships.
To implement this last step, the function takes a random point (p_i) and tries to restore its neighboring relationships by minimizing the error function, provided in eq (1). Then it proceed through a Breadth-First Search (BFS), inserting progressively in a queue all the neighbors of point p_i and repeating the procedure of error minimization.
This process is repeated until data points find a sort of equilibrium and their cumulative update is smaller than a given threshold. When this happens, the information along the axes to be discarded will be very small and we can safely project our data.

4 Comparisons

The Manifold Sculpting algorithm has been compared with other methods seen during the lectures. In particular Isomap, Diffusion Map, t-SNE, UMAP, LLE.
Also in the paper, the algorithm was compared with some of these techniques.

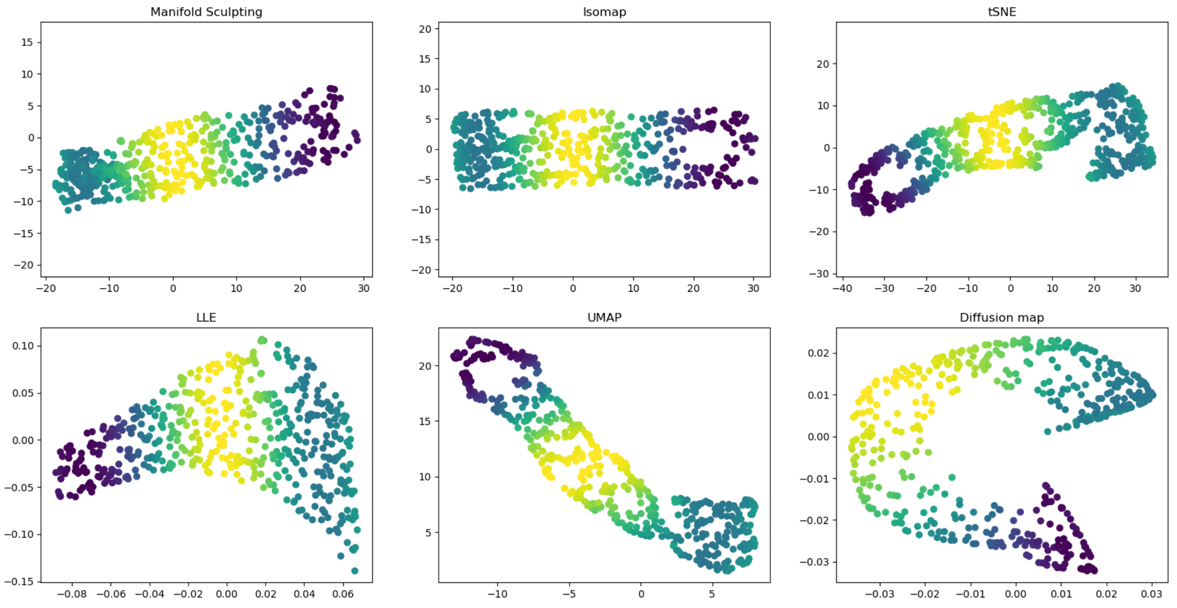


Figure 2: Results and comparison of different important manifold learning techniques on a Swiss Roll dataset of 500 data points, using a number of neighbors $k = 12$.

Isomap seems to achieve very good results, however, as stated in the paper, the main drawback of Isomap is that, being an MDS method, it needs to solve the eigenvalue-eigenvectors problem for a matrix of size $n \times n$, where n is the number of our data points.
Except for this, Isomap is a pretty straightforward algorithm which produces very good results. Manifold Sculpting, indeed, uses the same neighborhood relationships used by Isomap, but it tries to preserve them in a completely different way. It also adds colinear points to try to deal with curvatures, holes and lack of samples in some regions of the manifold.
The time complexity of Isomap is the following [2]:

- Search of nearest neighbor search. Isomap uses BallTree for efficient neighbor search. The cost is approximately: $O(d \cdot log(k) \cdot n \cdot log(n))$ for k nearest neighbors of n points in d dimensions.
- Shortest-path graph search; cost: $O(n^2(k + log(n))$ using Dijkstra's Algorithm.
- Partial eigenvalue decomposition. Cost: $O(d \cdot n^2)$

So, the final cost is:

$$O(d \cdot log(k) \cdot n \cdot log(n)) + O(n^2(k + log(n)) + O(d \cdot n^2)$$

The cost of Manifold Scultping is, instead, given by:

- Search of local relationships: $O(d \cdot log(k) \cdot n \cdot log(n) + k^2 \cdot n)$
- Iterative procedure to adjust local relationships, repeated for *max_iter* times:

  Scaling: $O(n)$

  Points adjustment: $O(n \cdot k \cdot d^3)$

So, the global cost is:

$$O(d \cdot log(k) \cdot n \cdot log(n) + k^2 \cdot n + max\_iter \cdot (n + n \cdot k \cdot d^3))$$

If we compare just the last part of the algorithm, which is what makes the main difference, we can see how Manifold Sculpting's cost should be smaller for a relative small number of initial dimensions and neighbors considered, however convergence is quite slow and so the main problem is given, indeed, by the number of iterations required.

The drawbacks of Manifold Scultping are various.

- It requires a lot of hyperparameters:

  - the number of nearest neighbors to be used k;

  - the scale factor σ;

  - the learning rate η;

  - the threshold to be used to know when to stop.
- It is dependent on the initialization.
- It is computationally expensive.

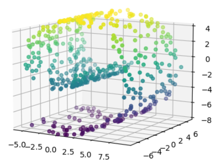
The main advantage and wonderful thing about manifold sculpting is that we see what is going on, stopping our algorithm at whatever point we want and looking at partial results, as it can be seen in figure 3.

5 Conclusions

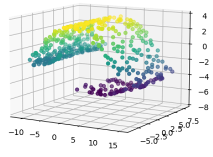
Manifold Sculpting is a very interesting Manifold Learning technique which produces well looking and self-explainable results.
However it has some critical points and, in particular, it needs to be properly tuned in order to obtain good results.

The author himself wrote on a forum, about a specific experiment he took: "Apparently, it took some experimentation before I was happy with the results."

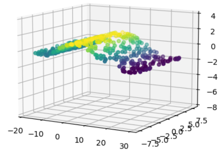
"Tangentially, if you're looking for an NLDR algorithm that just works in a variety of cases, frankly, I'd recommend looking into some of the neural network-based feature-learning methods. Manifold Sculpting can produce good results in specific cases, but it requires careful tuning and a lot of processing time. Ultimately, all of the NLDR methods that require neighbor-finding as their first step end up being too slow for general applications, too susceptible to problems with poorly-sampled manifolds, and unable to generalize effectively for out-of-band samples. The neural network approaches, by contrast, require no neighbor-finding step, and are designed to generalize." Discussion here



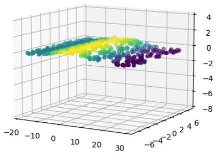
(a) Original dataset



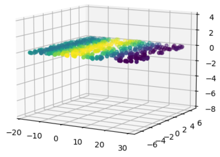
(b) 5 iterations



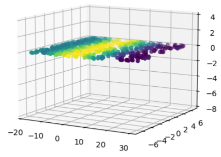
(c) 15 iterations



(d) 25 iterations



(e) 40 iterations



(f) 60 iterations

Figure 3: Unrolling Swiss Roll through Manifold Sculpting iterative procedure. This was obtained by an initial dataset of 500 data points, using a number of neighbors $k = 12$.

References

[1] Mike Gashler, Dan Ventura, and Tony Martinez. Iterative non-linear dimensionality reduction by manifold sculpting. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, page 513–520, Red Hook, NY, USA, 2007. Curran Associates Inc.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Manifold learning, 2011.