An Overview of Maximum Variance Unfolding

Sam Longenbach, Caleb Ralphs

November 2019

1 Introduction

Maximum variance unfolding (MVU), also known as semi-definite embedding, is a non-linear dimension reduction technique. It can be viewed as nonlinear generalization of the linear dimension reduction technique, principal component analysis (PCA). Like many dimension reduction algorithms, MVU looks to map high dimensional inputs to low dimensional outputs. The visual intuition behind MVU is this idea we connect neighboring points in the high dimensional space by rigid rods. The amount of connections or number of rods is a chosen parameter (k) we will discuss later on. With that said, the points connected to their k closest neighbors create this lattice-like structure. MVU can be seen as pulling this structure apart and flattening the structure without breaking or stretching the rigid rods. In other words, MVU looks to maximize the pairwise distances between input points subject to the constraints of maintaining the distances of these local rigid connections [1].

Dimension reduction techniques such as PCA, ISO-Map, and Kernel PCA rely on a predefined kernel that is then decomposed. As a reminder, a kernel is a function that computes similarities between high dimensional data inputs (X). In the case of PCA, these similarities are computed using a linear kernel resulting in the following kernel matrix $K = XTX$ where X is the input data. While in ISO-Map the geodesic distance matrix can be viewed as the kernel matrix. MVU operates differently in that we do not assume a kernel to start but instead set up an optimization problem to try to find an optimal kernel matrix. Once the kernel is found then we do eigenvalue decomposition on the kernel matrix. This optimization problem of finding the optimal kernel matrix is solved by using semi-definite programming. Below we will outline the optimization formulation and what we mean by the optimal kernel.

We find the optimal kernel matrix K with the following semi-definite program:

Maximize $\sum_i^n \sum_j^n ||y_i - y_j||_2^2)$ such that

1. $||y_i - y_j||_2^2) = ||x_i - x_j||_2^2)$ for all (i, j) with $\eta_{ij} = 1$
2. $\sum_i^n \sum_j^n K_{ij} = 0$
3. $K \succeq 0$

This convex optimization definition is in terms of x and y, the data points in the higher and lower dimensional spaces respectively, but we will redefine the problem in terms of Gramian matrices for the two spaces. This will allow us to find an optimal kernel matrix in terms of K, not in literal terms of distances between points.

2 MVU Algorithm

Maximum Variance Unfolding is structured as a semi-definite programming problem through the following definition of the optimization to find the kernel (Gramian) matrix K that maximizes the distances between all of the data except the points nearest to each other. To find the points nearest to one another, we must first construct the K-nearest neighbor (KNN) edge matrix E. For this part of the algorithm, k is parameterized and can be chosen based upon the rate of change of the gradient in the data, or in simpler, less accurate terms, the density of the data. As will be explained later, choosing a good k is crucial in the algorithm.

The edge matrix E, constructed from the KNN with k neighbors is of the following form:

$$E = \begin{bmatrix} \eta_{11} & \dots & \eta_{1n} \\ \dots & \dots & \dots \\ \eta_{n1} & \dots & \eta_{nn} \end{bmatrix}, \eta_{ij} \in 1, 0, \sum_j^n \eta_{ij} = k$$

Here, each entry in the edge matrix E takes on a value of wither 1 or 0, which will help define the first condition of the semidefinite program. The number of neighbors k dictates the connectivity of the graph, represented by the row sum of edge matrix E.

Also note that it will be helpful to know that the graph representing the edges between points in connected, as if disconnected, the objective function of maximizing distances between points will be unbounded and the distances between all of the disconnected points will go towards infinity. We can resolve this problem by checking the eigenvalues of the Laplacian of E, ensuring that all of the eigenvalues are positive [2].

Now that the local isometry conservation has been defined by the entries in E

with values of 1, we can rewrite that first constraint in terms of the Gramian, to start framing the problem as a semi-definite programming one [4].

$$\begin{aligned}
||x_i - x_j||_2^2 &= x_i^T x_i + x_j^T x_j - x_j^T x_i - x_i^T x_j \\
&= G_{ii} + G_{jj} - G_{ij} - G_{ji} \\
&= G_{ii} + G_{jj} - 2G_{ij}
\end{aligned}$$

We now have defined the distances between points as a series of operations with the respective Gramian matrix. This same distance representation can be applied to the kernel matrix K that we are going to be solving for. Recall, η represents entries in the edge matrix E

$$\begin{gathered}
||x_i - x_j||_2^2 = ||y_i - y_j||_2^2 \\
G_{ii} + G_{jj} - 2G_{ij} = K_{ii} + K_{jj} - 2K_{ij} \\
\Rightarrow \\
\eta_{ij}(G_{ii} + G_{jj} - 2G_{ij}) = \eta_{ij}(K_{ii} + K_{jj} - 2K_{ij})
\end{gathered}$$

The second constraint requires that the data be centered around the origin, which will allow us to rewrite the object function and frame the problem in terms of K, instead of the distances $||y_i - y_j||_2^2$. Centering the data makes the sum of all the points in the kernel matrix K equal to zero:

$$\begin{aligned}
\sum_i^n y_i = 0 &\iff (\sum_i^n y_i = 0)^T (\sum_i^n y_i = 0) \\
&\iff (\sum_i^n y_i = 0)^T (\sum_j^n y_j = 0) \\
&\iff \sum_i^n \sum_j^n y_i^T y_j = 0 \\
&\iff \sum_i^n \sum_j^n K_{ij} = 0
\end{aligned}$$

Now we can rewrite how we are maximizing the variance for those disconnected points, i.e., the zeros on the edge matrix E:

$$
\begin{aligned}
\sum_i^n \sum_j^n ||y_i - y_j||_2^2 &= \sum_i^n \sum_j^n (K_{ii} + K_{jj} - 2K_{ij}) \\
&= \sum_i^n \sum_j^n (K_{ii} + K_{jj}) \\
&= \sum_i^n \sum_j^n K_{ii} + \sum_i^n \sum_j^n K_{jj} \\
&= n \sum_i^n K_{ii} + n \sum_j^n K_{jj} \\
&= 2nTrace(K) \\
&\Rightarrow \\
max(\sum_i^n \sum_j^n ||y_i - y_j||_2^2) &= max(2nTrace(K)) \\
&= max(Trace(K))
\end{aligned}
$$

So we are now just maximizing the distance between points for all the points not connected in the edge matrix E, represented by maximizing the Trace of K.

A matrix is said to be a Positive Semi-Definite matrix if and only if

$$\sum_i^n \sum_j^n K(x_i, x_j) c_i c_j \geq 0$$

So we now go from the previously defined objective function and parameters:

Maximize $\sum_i^n \sum_j^n ||y_i - y_j||_2^2)$ such that

1. $||y_i - y_j||_2^2) = ||x_i - x_j||_2^2)$ for all (i, j) with $\eta_{ij} = 1$
2. $\sum_i^n \sum_j^n K_{ij} = 0$
3. $K \succeq 0$

And we arrive at the following semi-definite programming definition:

Maximize $Trace(K)$ such that

1. $\eta_{ij}(G_{ii} + G_{jj} - 2G_{ij}) = \eta_{ij}(K_{ii} + K_{jj} - 2K_{ij})$
2. $\sum_i^n \sum_j^n K_{ij} = 0$

3. $K \succeq 0$

3 Experiments with Landmark MVU

MVU requiring the use of semi-definite programming has the downside of being computationally expensive. Moreover, as the number of data instances n is increased the entries of the edge matrix grow at a rate of n^2 [5]. With a large edge matrix comes the problem of memory but also the issue of appropriately connecting the graph. Choosing a proper k in K-nearest neighbors is a balance. If k is too small, this will lead to a disconnected graph but if k is too large this may over constrain the learned representation of the data. For our experiments we found this parameter k to be extremely sensitive as varying it had significant affected our output.

In Python the only MVU implementation we could find was from the GitHub user buquicchiol [2]. When running our larger experiments we found success when using his Landmark MVU class. In addition to selecting k we also needed to select the number of landmarks L. His Landmark MVU version modifies the edge matrix in the following way. Like above we first construct the K-nearest neighbor (KNN) n by n edge matrix E for all n data instances. We then calculate and separate the top L most connected points in E. This can be done by taking the column sums of E which measures how many connections each point has. Once the points are separated E is then discarded.

Next for just the top L most connected points i.e. the landmarks, we construct the K-nearest neighbor (KNN) L by L edge matrix E_L. For the remaining $n-L$ data instances we construct an $(n-L)$ by L edge matrix E_l where we connect $n-L$ data instances to the landmarks. Lastly, we take all the connections in E_L and E_l and combine them into a new n by n edge matrix which is then passed to the semidefinite program solver like before. We found this modification of creating the edge matrix in this way lead to constraints in which the solver was able to find a feasible solution more readily.

We implemented Landmark MVU on the teapot data set which contains the 400 image sequence of a teapot being slowly rotated in a circle. Each RGB image in the data set had 76x101x3 = 23,028 pixels or dimensions. In figure 1 you can see a few of these raw images.

Below we tested varying the initial number of k nearest neighbors and varying L landmarks. For sake of comparison, we start by implementing PCA on the data. In figure 2 you can see an issue PCA has. PCA appears to struggle in discerning the difference between the handle and spout of the teapot. In the projection, these rotations are projected together.

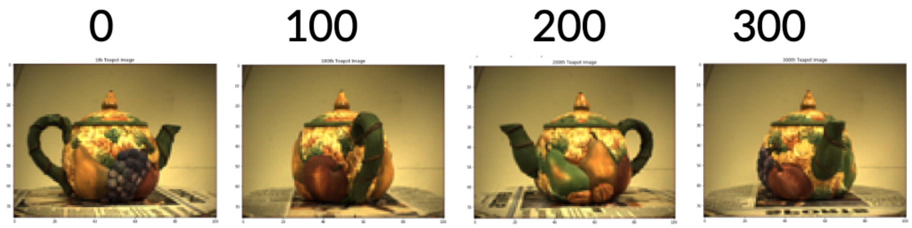

Figure 1: Ordered images of teapots throughout the counter clockwise rotation.

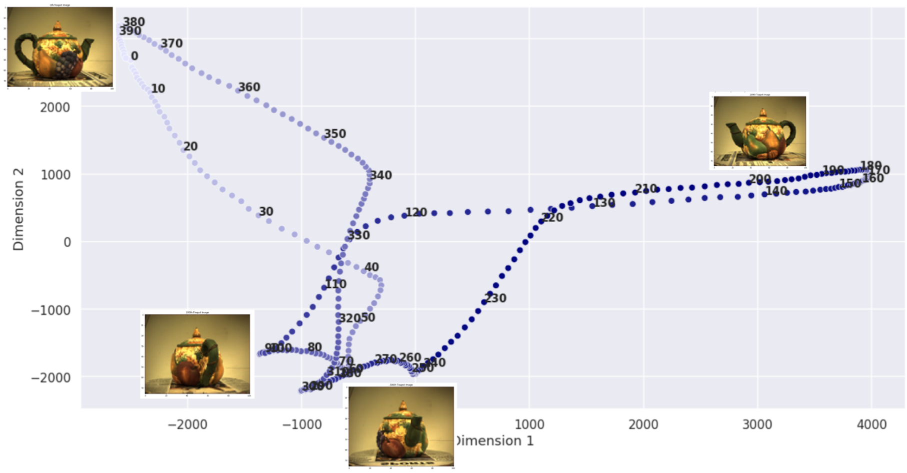

Figure 2: PCA projection of 400 teapot images.

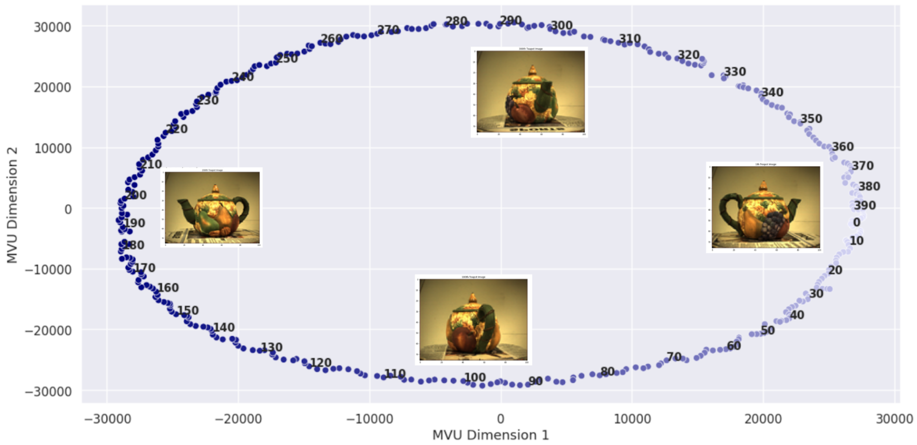

Figure 3: MVU projection of 400 teapot images with k=4, L=120.

When the correct parameters k and L are found, MVU excels at finding the underlying dimension of the teapot images that is a circle. In figure 3 when $k = 4$ and $L = 120$ appears to be the magic configuration. To test the sensitivity of L we fix $k = 4$ and decrease the number of landmarks. As you can see in figure 4 the low dimensional representation slowly starts to lose its circular shape. It is important to note that $L = 120$ was approaching the upper limit of the number of landmarks in this example. If $L = 150$, the edge matrix would be too connected and the solver would be unable to unfold the data and error out. With that said, we can also look at the sensitivity of k where we fix $L = 120$ and varying the initial number of nearest neighbors in creating E. In figure 5 it appears that MVU is extremely sensitive to k as increasing or decreasing k by one leads to results similar to PCA. Similar in the sense it struggles with discerning the handle and spout of the teapot.

4 Extension: Action Respecting Embedding

In this section, we present an extension of maximum variance unfolding called action respecting embedding (ARE) where it is assumed that the input data X is given a sequence of action labels [3]. Therefore action respecting embedding assumes there is an order to the input data X that must be respected. This unlike in MVU where the order of the n data points i.e. permuting the rows of the input data X won't affect the result of the low dimensional embedding. Action respecting embedding in a way is temporal MVU. For example, in ARE data could come from a mobile robot taking images where between each image some action is performed. The goal ARE is to find a low dimensional representation of this high dimensional set of ordered instances like MVU but

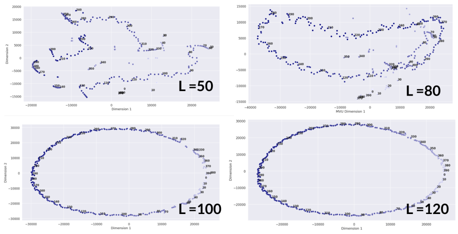

Figure 4: MVU projection of fixing k=4, and varying L.

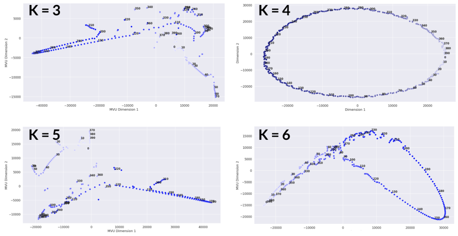

Figure 5: MVU projection of fixing L=120, and varying k.

furthermore imposes action respecting constraints. To formalize these action respecting constraints first let $X = x_1, x_2, x_3 ... x_n$ be the set of ordered images with associated discrete actions $a_1, a_2, a_3 ... a_{n-1}$.

The first constraint is a slight modification to the creation of the edge matrix E or neighborhood graph in MVU. As a reminder, MVU constructs E from using KNN with k neighbors. As previously discussed choosing a proper k is a balance between being large enough resulting in a connected graph but not too large to over constrain the learned representation. However, since we have additional information that certain pairs of instances are associated with actions, ARE looks to build a more informed edge matrix i.e. non-uniform neighborhood graph. The idea presented in the paper is that each instance has a circle with some radius ϵ such that ϵ is large enough to encompass all data points connected by an action. In turn, an edge is created between two data instances only if they reside in each other's ϵ circle. Lastly, T is introduced as a parameter to increase the connectivity of E. T is the action window size that furthermore requires the circles around data instances to be large enough to encompass T actions. Then data instances within T actions would be connected as opposed to just a single action.

Therefore given T this constraint is denoted:

$$\begin{aligned}\eta_{ij} = 1 \iff \exists h, k \text{ s.t.} \\ \mid h - i \mid < T, \mid k - j \mid < T, \\ ||x_i - x_h||_2^2 > ||x_i - x_j||_2^2 \text{ and } ||x_j - x_k||_2^2 > ||x_i - x_j||_2^2\end{aligned}$$

The second constraint of ARE imposes the action respecting requirement. It is important to note that having $a_1, a_2, a_3...a_{n-1}$ provides information even if the actions themselves lack concrete interpretation. The paper uses the example of a camera taking pictures as it moves left then right. We would like the learned embedding of these data instances to be represented as translations. Furthermore, it is required that all actions should be composed of translations and rotations. This restriction is imposed as these so that actions are distance preserving. This is important as all actions must be distance preserving in the low dimensional learned embedding. Let f_a denote a distance preserving action. Hence for any two data instances in low dimensional space y_i and y_j it must be the case that:

$$\forall i, j \, ||y_i - y_j||_2^2 = ||f_a(y_i) - f_a(y_j)||_2^2$$

In words, the distance between two points before an action must be equivalent to the distance between two points after an action. With this we can consider the case where in our sequence of actions $a_1, a_2, a_3...a_{n-1}$ if $a_i = a_j$. If $a_i = a_j$, it holds that $f_a(y_i) = y_{i+1}$ and $f_a(y_j) = y_{j+1}$ so by substitution we now have:

$$||y_i - y_j||_2^2 = ||y_{i+1} - y_{j+1}||_2^2$$

Therefore using similar logic to how we framed the first MVU constraint in terms of the kernel matrix K we impose constraints on the kernel matrix such that:

$$\forall i, j \quad a_i = a_j \implies K_{ii} + K_{jj} - 2K_{ij} = K_{(i+1)(i+1)} + K_{(j+1)(j+1)} - 2K_{(i+1)(j+1)}$$

In total, Action Respecting Embedding modifies the creation of the edge matrix E. Instead of simply running and constructing the K-nearest neighbor graphs, it takes into account that pairs of instances are associated with actions. Therefore assuming E is created following the constraint discussed above and we impose that actions are distance preserving in the low dimensional space, we can frame ARE as the following semi-definite program:

Maximize $Trace(K)$ such that

1. $\eta_{ij}(G_{ii} + G_{jj} - 2G_{ij}) \geq \eta_{ij}(K_{ii} + K_{jj} - 2K_{ij})$
2. $\sum_i^n \sum_j^n K_{ij} = 0$
3. $K \succeq 0$
4. $\forall i, j \quad a_i = a_j \implies$
   $K_{ii} + K_{jj} - 2K_{ij} = K_{(i+1)(i+1)} + K_{(j+1)(j+1)} - 2K_{(i+1)(j+1)}$

Lastly, do note that equality in 1. has been changed to an upper bound. This is to ensure feasibility by allowing the zero matrix as a solution.

5 Summary

Like many dimension reduction algorithms, MVU looks to map high dimensional inputs to low dimensional outputs. MVU is special in comparison to other well known DR techniques (Kernel PCA, ISO-map, ..) as a kernel is not predefined in MVU. Instead, MVU sets up an optimization problem to find an optimal kernel which is then solved using semi-definite programming. MVU looks to maximize the pairwise distances between input points subject to the constraints of maintaining the distances of these local rigid connections. These local rigid connections are defined by creating an edge matrix using K-nearest neighbors. On the upside, MVU perfectly conserves these rigid connections or local isometries in the low dimensional embedding. Furthermore, these constraints can be relaxed in an attempt to handle noisy data. Once a kernel matrix is optimally found, the top eigenvalues indicate the underlying dimensionality of the data. MVU under the right initial parameters performs well when it comes to discovering underlying dimensionality. This is shown in the example of the teapot data set results. MVU does a great job of distinguishing the differences between the handle and spout even though pixel-wise these two orientations are very similar.

With that said, unfortunately, MVU also has some downsides. First off semi-definite embedding is computationally expensive and in practice, MVU can only handle at most a few thousand data instances. To avoid creating an edge matrix that leads to a disconnect or over connects the data, we run our experiments using Landmark MVU. Even with this implementation, we saw that the embedding is still extremely sensitive to the nearest neighbors parameter k. The computation time of semi-definite programming coupled with the sensitivity of parameters k and the number of landmarks L resulted in a cumbersome process to find an ideal embedding in the case of the teapot example.

References

[1] (65) Ali Ghodsi, Lec 7: MVU, Action Respecting Embedding, Supervised PCA - YouTube.

[2] MaximumVarianceUnfolding/MVU.py at master · buquicchiol/MaximumVarianceUnfolding.

[3] Michael Bowling, Bowling@ualberta Ca, Ali Ghodsi, and Dana Wilkinson. Action Respecting Embedding. Technical report.

[4] Ryan Tibshirani Geoff Gordon. Maximum Variance Unfolding, 2012.

[5] Jianzhong Wang and Jianzhong Wang. Maximum Variance Unfolding. In *Geometric Structure of High-Dimensional Data and Dimensionality Reduction*, pages 181–202. Springer Berlin Heidelberg, 2012.