

A Survey of Dimensionality Reduction Techniques

RACHEL LEVANGER

This note set gives a short summary of a collection of commonly-used linear and non-linear dimensionality reduction techniques. When applicable, we include a comment on how the technique can be applied to analyzing the geometry of a point cloud in the space of persistence diagrams.

Contents

1	Linear Dimensionality Reduction (LDR) Techniques	2
1.1	Principal Component Analysis (PCA)	2
1.2	Multidimensional scaling (MDS)	3
2	Non-linear Dimensionality Reduction (NLDR) Techniques	3
2.1	Kernel PCA	3
2.2	Locally-linear Embedding (LLE)	5
2.3	Hessian Locally-linear Embedding (Hessian LLE)	8
2.4	Isomap	8
2.5	Laplacian Eigenmaps	8
2.6	Diffusion Maps	8

1 Linear Dimensionality Reduction (LDR) Techniques

1.1 Principal Component Analysis (PCA)

PCA is a tool used to describe a data set in terms of its extrinsic (linear) directions of highest variance, and does so in a way that the data is uncorrelated with respect to these directions. Consider a set S of m points in \mathbb{R}^n . Then S can be represented as an $m \times n$ matrix X . Call $\mu_X = (\mu_1, \dots, \mu_n)$ the center of the data set, or the average of each of the columns of X . The general sequence of steps to perform a PCA on X is as follows:

1. Traditionally, compute the covariance matrix Σ associated to the data set X , where

$$\Sigma_{ij} = \text{cov}(X_{*i}, X_{*j}) = (1/m) \sum_{k=1}^m (X_{ki} - \mu_i)(X_{kj} - \mu_j).$$

Notice that this is just the dot product of the mean-centered column vectors of the data matrix divided by the number of points in S . It is also possible to use $\Sigma = X^T X$ or the correlation matrix, as these are all essentially functions of the dot product, they are real and symmetric, and so diagonalization yields an orthogonal set of eigenvectors. Using $\Sigma = X^T X$ is essentially the computation of the Singular Value Decomposition (SVD) of X as it relates to finding the directions of highest variance.

2. Since Σ is real and symmetric, we can diagonalize Σ via the matrices Φ and Λ , where Φ is orthogonal and Λ is diagonal. The matrix Φ has columns that form the orthonormal basis of eigenvectors of Σ , Λ gives the corresponding eigenvalues, and the equation $\Sigma = \Phi \Lambda \Phi^T$ holds.
3. The magnitude of the eigenvalues in Λ provide insight into the number of dimensions along which the data is organized. Furthermore, since the eigenvectors are orthogonal, the covariance along any two transformed basis vectors are zero (their dot product is zero).

sources:

<https://stat.duke.edu/~sayan/Sta613/2015/lec/IDAPILecture15.pdf>

<http://infolab.stanford.edu/~ullman/mmds/ch11.pdf>

1.2 Multidimensional scaling (MDS)

2 Non-linear Dimensionality Reduction (NLDR) Techniques

2.1 Kernel PCA

The idea is to take the original data set in \mathbb{R}^D and map it (non-linearly) to a feature space via a map $\Phi : \mathbb{R}^D \rightarrow F$, and then do PCA in the space F . Recall that PCA relies heavily on the use of the dot product, and so the goal is to perform dot products in feature space. However, explicit computation of dot products in feature space, after performing a non-linear mapping, is not required. Instead, the computation of a kernel function on the input space is used as a proxy.

Reformulation of PCA to a problem of dot products:

Given a collection of mean-centered data points $X = \{x_1, \dots, x_N\}$ so that $\sum_{i=1}^N x_i = 0$, PCA boils down to a problem of finding eigenvalues and eigenvectors corresponding to the covariance matrix

$$C = \frac{1}{N} \sum_{i=1}^N x_i x_i^T.$$

Thus, one must find pairs λ, v such that

$$\lambda v = Cv = \frac{1}{N} \sum_{i=1}^N (x_i x_i^T) v = \frac{1}{N} \sum_{i=1}^N (x_i \cdot v) x_i.$$

From this equation, it is clear that each eigenvector v lies in the span of X , and so the above equation is equivalent to solving

$$(1) \quad \lambda(x_i \cdot v) = (x_i \cdot Cv) \quad \text{for all } i = 1, \dots, N,$$

and we know there must exist a vector $\alpha = [\alpha_1 \dots \alpha_N]$ such that $v = \sum_{i=1}^N \alpha_i x_i$. Thus, Equation (1) becomes

$$(2) \quad \lambda \sum_{j=1}^N \alpha_j (x_i \cdot x_j) = \frac{1}{N} \sum_{j=1}^N \alpha_j \left(x_i \cdot \sum_{k=1}^N x_k \right) (x_k \cdot x_j)$$

for all $i = 1, \dots, N$. Thus, defining an $N \times N$ matrix K by $K_{ij} = (x_i \cdot x_j)$, we can reformulate Equation (2) as

$$(3) \quad N\lambda K\alpha = K^2\alpha.$$

Since K is symmetric, its set of eigenvectors spans the entire space, and so Equation (3) implies the solutions to $N\lambda\alpha = K\alpha$ give all vectors α solving Equation (3). Furthermore, since K is positive semidefinite, its eigenvalues are nonnegative, and these

eigenvalues will be the values $N\lambda$ solving the equation $N\lambda\alpha = K\alpha$. Diagonalizing K yields eigenvalues $\lambda_1 \leq \dots \leq \lambda_N$ and a complete set of corresponding eigenvectors $\alpha^1, \dots, \alpha^N$. Say that λ_p is the least non-zero eigenvalue. We then normalize the α_i for $i = p, \dots, N$ according to the rule that the associated eigenvectors of C yield $(v^i \cdot v^i) = 1$ for $i = p, \dots, N$. To restate this in terms of a normalization on the α^i , we see that

$$\begin{aligned}
 1 &= (v^i \cdot v^i) \\
 &= \left(\sum_{j=1}^N \alpha_j^i x_j \cdot \sum_{k=1}^N \alpha_k^i x_k \right) \\
 &= \sum_{j,k=1}^N \alpha_j^i \alpha_k^i (x_j \cdot x_k) \\
 &= \sum_{j,k=1}^N \alpha_j^i \alpha_k^i K_{jk} \\
 &= (\alpha^i \cdot K \alpha^i) \\
 &= \lambda^i (\alpha^i \cdot \alpha^i)
 \end{aligned}$$

for all $i = p, \dots, N$. Recall that for PCA, we need to perform projections of the $x \in X$ onto the eigenvectors v^i corresponding to C . Thus, we can call

$$(v^i \cdot x) = \sum_{j=1}^N \alpha_j^i (x_j \cdot x)$$

the principal components. Notice that this formulation is given entirely in terms of dot products of the points in X .

Adding a nonlinear kernel:

The prior discussion took place in an arbitrary vector space. As mentioned in the introduction to this section, the goal is to map the set $X = \{x_1, \dots, x_N\}$ to a collection of points in feature space $\Phi(X) = \{\Phi(x_1), \dots, \Phi(x_N)\}$ and then perform PCA. Hence, the above discussion goes through to feature space and we get the principal components

$$(v^i \cdot \Phi(x)) = \sum_{j=1}^N \alpha_j^i (\Phi(x_j) \cdot \Phi(x))$$

corresponding to Φ in feature space. The key is in the definition of the matrix $K_{ij} = (\Phi(x_i) \cdot \Phi(x_j))$. Here, instead of using an actual dot product in feature space, we use a kernel representation $k(x, y) = (\Phi(x) \cdot \Phi(y))$. The primary result used is that certain kernel functions lead to constructions of maps Φ under which k behaves as a

dot product in that space. Hence, a choice of a kernel map k implicitly chooses for us a mapping Φ and a corresponding feature space F in which we perform the PCA. This leads us to the

Algorithm:

Given a set of points $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^D$, kernel PCA proceeds as follows:

1. Choose a suitable kernel function $k(x, y)$ and compute the associated matrix of dot products $K_{ij} = k(x_i, x_j)$.
2. Diagonalize K to solve for the eigenvalues $\lambda_1 \leq \dots \leq \lambda_N$ and associated eigenvectors $\alpha^1, \dots, \alpha^N$, and normalize them so that $1 = \lambda_i(\alpha^i \cdot \alpha^i)$.
3. Get a projection function

$$(kPCA)_i(x) = \sum_{j=1}^N \alpha_j^i k(x_j, x)$$

onto the i^{th} principal component of x in the feature space. For suitably chosen kernels, this corresponds to PCA in some high-dimensional feature space.

Discussion about kernels:

How does one go about choosing an appropriate kernel function k ?

source:

http://www.face-rec.org/algorithms/Kernel/kernelPCA_scholkopf.pdf

2.2 Locally-linear Embedding (LLE)

This technique works best when the points in your dataset are assumed to have densely sampled a portion of a manifold that can be ‘flattened out’ in some sense. For instance, the swiss roll is a two-dimensional manifold with boundary that can be flattened out into a portion of a two-dimensional plane, while a torus cannot be flattened out as such and would not be a good candidate for this method if one desires to perform a projection into two-dimensional space. Locally-linear embedding (LLE) also rests on the assumption that points that are nearest to a target point lie on the same portion of the manifold as the target point, i.e. the sampling density is much smaller than the weak feature size of the underlying manifold.

Let $X = \{x_1, \dots, x_N\}$ be a set of points in \mathbb{R}^D , $K \in \mathbb{N}$, and $d < D$ be the desired embedding dimension. The general idea of performing a LLE is as follows:

First, a weight matrix W is constructed that minimizes a certain error function. The error of the weight assignment can be computed as

$$\mathcal{E}(W) = \sum_i \left| x_i - \sum_j W_{ij} x_j \right|^2.$$

Thus, when the error is minimized, the distance from x_i to a weighted sum of its neighbors is minimized. We require that $W_{ij} = 0$ if x_j is not a neighbor of x_i , and $\sum_j W_{ij} = 1$, so that we are taking a weighted sum of vectors that are all neighbors of x_i . How is the weight matrix W constructed? For a data point x , call its set of K nearest neighbors $\{\eta_j\}_{j=1}^K$ and $\{w_j\}_{j=1}^K$ the set of weights associated to each. The error for the weight assignment of a single point x can be expressed as

$$\varepsilon(x) = \left| x - \sum_{j=1}^K w_j \eta_j \right|^2 = \left| x \sum_{j=1}^K w_j - \sum_{j=1}^K w_j \eta_j \right|^2 = \left| \sum_{j=1}^K w_j (x - \eta_j) \right|^2.$$

Now, let $C_{jk} = (x - \eta_j) \cdot (x - \eta_k)$, the covariance of the neighbors of x to the point x itself. Then since $|x|^2 = x \cdot x$, it follows that

$$\begin{aligned} \varepsilon(x) &= \left(\sum_{j=1}^K w_j (x - \eta_j) \right) \cdot \left(\sum_{k=1}^K w_k (x - \eta_k) \right) \\ &= \sum_{j=1}^K \sum_{k=1}^K w_j w_k (x - \eta_j) \cdot (x - \eta_k) \\ &= \sum_{j,k} w_j w_k C_{jk}. \end{aligned}$$

To find the vectors w_j , it is sufficient to solve the system of equations $\sum_j C_{jk} w_k = 1$, then rescale the w_j so that they sum to 1.

Next, the embedded dataset $Y = \{y_1, \dots, y_N\}$ must be constructed. Essentially, this dataset is the one that minimizes the cost function

$$\Phi(Y) = \sum_i \left| y_i - \sum_j W_{ij} y_j \right|^2$$

after the weight matrix W has already been computed from X . Since the weights have already been selected so that they represent a good relationship between the distance between a data point and its neighbors, minimizing Φ via an embedded dataset Y stands a good chance of maintaining these neighbor relationships. How are the points

Y found? Set $M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki}W_{kj}$, where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. Then the cost function Φ can be restated as

$$\Phi(Y) = \sum_{i,j} M_{ij}(y_i \cdot y_j).$$

Two assumptions are added to make the optimization possible:

- (i) The points Y are centered at the origin, so that $\sum_{i=1}^N y_i = 0$.
- (ii) The covariance matrix of Y is the $d \times d$ identity matrix, so that $\frac{1}{N} \sum y_i y_i^T = I_d$.

Finally, the bottom $d + 1$ eigenvectors of $M = (I - W)^T(I - W)$ (with the exception of the eigenvector corresponding to the smallest eigenvalue) form the embedding vectors.

Algorithm:

1. *Find the K nearest neighbors of each point.* Compute the distance matrix of the data set X . Let $N_K : X \rightarrow \mathcal{P}(X)$ be the function that assigns to each x_i its K nearest neighbors, $N_K(x_i)$, which can be gleaned from the distance matrix.
2. *Solve for the reconstruction weights.*
 - (a) For each $x_i \in X$, create a matrix Z_i with a column for each element in $N_K(x_i)$. Subtract x_i from each column in Z (center at x_i).
 - (b) Set $C_i = Z_i^T Z_i$ (K times the local covariance), and solve for a vector w_i such that $C_i w_i = [1]$, where $[1]$ is a column vector of all ones.
 - (c) Populate the i^{th} row of the weight matrix W as follows. If $x_j \notin N_K(x_i)$, set $W_{ij} = 0$. If $x_j \in N_K(x_i)$ then set $W_{ij} = w_j / (w \cdot [1])$, where w_j is the entry in w corresponding to the neighbor x_j .
3. *Compute the embedding coordinates.* Set $M = (I - W)^T(I - W)$. Find the $d + 1$ eigenvectors corresponding to the smallest eigenvalues of M . Set the q^{th} row of Y to be the eigenvector corresponding to the $(q + 1)^{th}$ -smallest eigenvalue. (Thus, the eigenvector corresponding to the smallest eigenvalue of M is not used in Y).

Notes for the space of persistence diagrams:

The algorithm as presented requires that we begin with points in linear space, as solving for the weight matrix W involves performing linear algebra on the original dataset. However, it is possible to assign a weight matrix W that does not optimize the error function as stated, but instead assigns weights based on a rule or function. The second part of the algorithm optimizes the embedding vectors with respect to a cost function, regardless of how that weight matrix was constructed, and so the algorithm can continue at step (3) as stated. Preliminary tests show that using LLE in this manner for periodic

orbits in the space of persistence diagrams does reconstruct the loop in projected space, although the local manner in which the embedding is constructed does not faithfully represent any distortions in the loop such as pinches or twists, and so more complicated dynamics cannot be recovered by this embedding.

sources:

<https://www.cs.nyu.edu/~roweis/lle/algorithm.html>

2.3 Hessian Locally-linear Embedding (Hessian LLE)

2.4 Isomap

2.5 Laplacian Eigenmaps

2.6 Diffusion Maps

rachel@math.rutgers.edu