APML

Dr. Matan Gavish

Fall 2019

Lecture 2: Manifold Learning (II)

Updated January 2, 2020

האוניברסיטה העברית בירושלים
THE HEBREW UNIVERSITY OF JERUSALEM

Lecture 2: Manifold Learning (II)

1. Locally Linear Embedding - LLE
2. Diffusion Maps
3. Code

Reminder: Last week's lecture

Goals:

- Understand math foundation of popular data analysis algorithms
- Called "manifold learning" or "nonlinear dimension reduction"
- These methods are used for
    - data visualization
    - data organization
    - clustering
    - preprocessing before standard ML algorithms (classification, regression, ranking, etc)

Meditation (I)

Many people don't understand the difference between PCA and MDS. Don't be one of them. Hint: In PCA we get the data points $\mathbf{x}_i$ and diagonalize a p-by-p matrix. In MDS we **only** observe the distances, not the points, and diagonalize an n-by-n matrix.

Meditation (II)

Think long and hard about why **diagonalization** appears in both PCA and MDS. What's so magical about diagonalization?

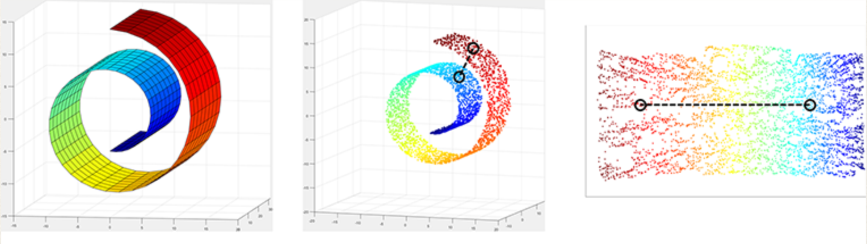
Meditation (III)

Suppose that there are points on a grid in $\mathbb{R}^p$ and we are interested in a function $f : \mathbb{R}^p \to \mathbb{R}$. We are only given $f(\mathbf{x}_i) - f(\mathbf{x}_j)$ for nearby points $\mathbf{x}_i, \mathbf{x}_j$. Can we recover f? Yes we can - this is called *integration*. In manifold learning we are recovering a global shape from local difference affinity (only use local diffeerences!). The integration machine is *diagonalization*.

Recall: Goal

- **Unsupervised** methods to recover intrinsic coordinates form high-dimensional data.
- Input: data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$
- Output: $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^d$
- where small distances are preserved: $||\mathbf{x}_i - \mathbf{x}_j|| \approx ||\mathbf{y}_i - \mathbf{y}_j||$ if $||\mathbf{x}_i - \mathbf{x}_j|| \ll 1$
- and where large distances $||\mathbf{y}_i - \mathbf{y}_j||$ now correspond to "intrinsic" distances, or **on the manifold**

Recall: want to fix this


(a) (b) (c)

Source: Wang et al, Head pose estimation, in: Manifolds: current research areas, 2017

Locally Linear Embedding - LLE

LLE

- The first method proposed
- In LLE we do a different linear dimensionality reduction at each point $\mathbf{x}_i$, and then combine then with minimal discrepancy.
- LLE is “low tech” - it really tries to follow the manifold idea verbatim.

Original LLE paper

Nonlinear Dimensionality Reduction by Locally Linear Embedding

Sam T. Roweis[1] and Lawrence K. Saul[2]

Many areas of science depend on exploratory data analysis and visualization. The need to analyze large amounts of multivariate data raises the fundamental problem of dimensionality reduction: how to discover compact representations of high-dimensional data. Here, we introduce locally linear embedding (LLE), an unsupervised learning algorithm that computes low-dimensional, neighborhood-preserving embeddings of high-dimensional inputs. Unlike clustering methods for local dimensionality reduction, LLE maps its inputs into a single global coordinate system of lower dimensionality, and its optimizations do not involve local minima. By exploiting the local symmetries of linear reconstructions, LLE is able to learn the global structure of nonlinear manifolds, such as those generated by images of faces or documents of text.

How do we judge similarity? Our mental representations of the world are formed by processing large numbers of sensory inputs—including, for example, the pixel intensities of images, the power spectra of sounds, and the joint angles of articulated bodies. While complex stimuli of this form can be represented by points in a high-dimensional vector space, they typically have a much more compact description. Coherent structure in the world leads to strong correlations between inputs (such as between neighboring pixels in images), generating observations that lie on or close to a smooth low-dimensional manifold. To compare and classify such observations—in effect, to reason about the world—depends crucially on modeling the nonlinear geometry of these low-dimensional manifolds.

Scientists interested in exploratory analysis or visualization of multivariate data (*1*) face a similar problem in dimensionality reduction. The problem, as illustrated in Fig. 1, involves mapping high-dimensional inputs into a low-dimensional "description" space with as many coordinates as observed modes of variability. Previous approaches to this problem, based on multidimensional scaling (MDS) (*2*), have computed embeddings that attempt to preserve pairwise distances [or generalized disparities (*3*)] between data points; these distances are measured along straight lines or, in more sophisticated usages of MDS such as Isomap (*4*), along shortest paths confined to the manifold of observed inputs. Here, we take a different approach, called locally linear embedding (LLE), that eliminates the need to estimate pairwise distances between widely separated data points. Unlike previous methods, LLE recovers global nonlinear structure from locally linear fits.

The LLE algorithm, summarized in Fig. 2, is based on simple geometric intuitions. Suppose the data consist of N real-valued vectors $\vec{X}_i$, each of dimensionality D, sampled from some underlying manifold. Provided there is sufficient data (such that the manifold is well-sampled), we expect each data point and its neighbors to lie on or close to a locally linear patch of the manifold. We characterize the local geometry of these patches by linear coefficients that reconstruct each data point from its neighbors. Reconstruction errors are measured by the cost function

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \Sigma_j W_{ij} \vec{X}_j \right|^2 \qquad (1)$$

which adds up the squared distances between all the data points and their reconstructions. The weights W_{ij} summarize the contribution of the jth data point to the ith reconstruction. To compute the weights W_{ij}, we minimize the cost

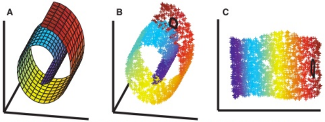


Fig. 1. The problem of nonlinear dimensionality reduction, as illustrated (*10*) for three-dimensional data (B) sampled from a two-dimensional manifold (**A**). An unsupervised learning algorithm must discover the global internal coordinates of the manifold without signals that explicitly indicate how the data should be embedded in two dimensions. The color coding illustrates the neighborhood-preserving mapping discovered by LLE; black outlines in (**B**) and (**C**) show the neighborhood of a single point. Unlike LLE, projections of the data by principal component analysis (PCA) (*28*) or classical MDS (*2*) map faraway data points to nearby points in the plane, failing to identify the underlying structure of the manifold. Note that mixture models for local dimensionality reduction (*29*), which cluster the data and perform PCA within each cluster, do not address the problem considered here: namely, how to map high-dimensional data into a single global coordinate system of lower dimensionality.

[1]Gatsby Computational Neuroscience Unit, University College London, 17 Queen Square, London WC1N 3AR, UK. [2]AT&T Lab—Research, 180 Park Avenue, Florham Park, NJ 07932, USA.

E-mail: roweis@gatsby.ucl.ac.uk (S.T.R.); lsaul@research.att.com (L.K.S.)

Source: `sciencemag.org`

LLE overview

1. Build a neighborhood for each point $\mathbf{x}_i$, consisting of points that are close-by in Euclidean distance on $\mathbb{R}^p$.
2. create a **linear** dimension reduction in each point separately
3. find low-dimensional coordinates continuously reconstructed from these weights

LLE steps

LLE step 1. For each point $\mathbf{x}_i$ ($i = 1, \ldots, n$) find its k nearest neighbors.

- k is a tuning parameter
- (How to find nearest neighbors??)

LLE steps

LLE step 2. Find weight matrix W that minimizes the **residual sum of squares** for reconstructing $\mathbf{x}_i$ from its nearest neighbors

$$RSS(W) := \sum_{i=1}^{n} \left\| \mathbf{x}_i - \sum_{j \neq i} W_{i,j} \mathbf{x}_j \right\|^2$$

where $W_{i,j} = 0$ unless $\mathbf{x}_j$ is of the k nearest neighbors of $\mathbf{x}_i$, and where $\sum_j W_{i,j} = 1$.

- (another name for minimizing RSS?)
- Note that if we also add the constrain $W_{i,j} \geq 0$, then W is an n-by-n **stochastic matrix**
- In other words it's a **Markov transition matrix** (Pagerank anyone?)

LLE steps

LLE step 3. Find the coordinates Y which minimize the reconstruction error using weights

$$\Phi(Y) = \sum_{i=1}^{n} \left\| \mathbf{y}_i - \sum_{i \neq j} W_{i,j} \mathbf{y}_j \right\|^2$$

subject to $\sum_i Y_{i,j} = 0$ for each j, and $Y^\top Y = I$.

- hah?
- we'll explain this soon

LLE - discussion

LLE step 1.

- Almost any manifold learning algorithm starts by finding *k* NN for each data point
- We could instead find, for each $\mathbf{x}_i$, all the points in the ε-ball around $\mathbf{x}_i$
- By using *k*-NN we effectively put smaller ε where datapoints are dense and larger ε where datapoints are further apart
- In other words: fine-grained view where we have lots of data, and coarse-grained view where we have little data
- Experiment for yourself - what happens to the algorithm as we vary *k*? what happens if we use ε-balls instead, and what's the effect of ε then?

A note on Nearest Neighbor search

- Again, most manifold learning algorithm start with *k*-NN search (since only small distances are dependable)
- What's the worst-case complexity of running exact *k*-NN search on *n* points in $\mathbb{R}^p$?
- A popular algorithm is **k-d tree**. It creates a data structure that's fast to query. You should be familiar with it. Lots of open source implementations.
- There's an entire industry of **approximate** *k*-NN search. One recent idea I like goes like so:
  - Random-project the *n* points to $\mathbb{R}^d$, $d \ll p$. (random projection is a dim-reduction method we didn't discuss last time)
  - Do exact *k*-NN search in $\mathbb{R}^d$
  - Repeat several times and do a clever averaging of the results
  - Can be parallelized to be deadly efficient.
  - See [Jones, Osipov, Rokhlin, PNAS 108(38) 2011]

LLE - discussion

LLE step 2.

- Why do we minimize

$$RSS(W) := \sum_{i=1}^{n} \left\| \mathbf{x}_i - \sum_{j \neq i} W_{i,j} \mathbf{x}_j \right\|^2$$

- Pretend the data is **exactly** linear in k-NN of $\mathbf{x}_i$
- Then $\mathbf{x}_i = \sum_j W_{i,j} \mathbf{x}_j$
- Now pretend the data is on a low-dimensional manifold. Convince yourself that the weights describe the structure of the manifold.

LLE - discussion (cont.)

LLE step 2.

- Note that we minimize *RSS* for each $\mathbf{x}_i$ - **separately**
- Fix *i*. Let's replace x_i by y and $W_{i,j}$ by β_j. For each *i* separately, we minimize

$$RSS = \left\| \mathbf{y} - \sum_{j \in NN(i)} \beta_j \mathbf{x}_j \right\|^2$$

- Looks familiar? **Without** the constrain $\sum \beta_j = 1$ this would mean we seek to express **y** as a **linear** combination of $\{\mathbf{x}_j\}$
- This is known as **linear regression**
- **With** the constrain $\sum \beta_j = 1$ this would mean we seek to express **y** as an **affine** combination of $\{\mathbf{x}_j\}$

LLE - discussion

LLE step 3.

- Why minimize $\Phi(Y) = \sum_{i=1}^{n} \left|\left|\mathbf{y}_i - \sum_{i \neq j} W_{i,j}\mathbf{y}_j\right|\right|^2$?
- Recall that when data is on linear subspace (simplest manifold) then intrinsic coordinates are coordinates in basis of subspace
- So observe that when data is on linear subspace, weights W obtained from $\mathbf{x}_1, \ldots, \mathbf{x}_n$ should equal the weights for intrinsic coordinates $\mathbf{y}_1, \ldots, \mathbf{y}_n$
- The idea behind LLE is that when data is on manifold and curvature is small (so k-NN all live in a small neighborhood, so roughly on a linear subspace)
- Then intrinsic coordinates $\mathbf{y}$ are those that best fit the weights that we found from the original data $\mathbf{x}$

Homework

1. LLE is invariant under **translation**: the algorithm yields the same result on $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and on $\mathbf{x}_1 + \mathbf{c}, \ldots, \mathbf{x}_n + \mathbf{c}$, for any $\mathbf{c}$
2. LLE is invariant under **rotation**: the algorithm yields the same result on $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and on $O \cdot \mathbf{x}_1, \ldots, O \cdot \mathbf{x}_n$, for any orthogonal matrix O
3. Why do we add the constrains $\sum_i Y_{i,j} = 0$ and $Y^\top Y = I$ to the last step of LLE?

LLE - computation

LLE step 2.

- How to find the weights W?
- Solve for each $RSS_i = \left|\left|\mathbf{x}_i - \sum_{j \neq i} W_{i,j} \mathbf{x}_j\right|\right|^2$ separately. Fix i.
- By translation invariance, $RSS_i = ||W_{i,j} \mathbf{z}_j||^2$ where $\mathbf{z}_j = \mathbf{x}_j - \mathbf{x}_i$.
- Let G be the k-by-k matrix consisting of all inner products of neighbors and let $\mathbf{w}_i$ be the k-vector of weights of neighbors of $\mathbf{x}_i$.
- Then $RSS_i = \mathbf{w}_i^\top G \mathbf{w}_i$. Note that G only depends on the data. Need to minimize RSS_i w.r.t $\mathbf{w}_i$ under the constrain $\sum_i (w_i)_j = 1$.
- To handle the constrain, introduce a Lagrange multiplier λ and obtain the Lagrangian $L(\mathbf{w}_i, \lambda) = \mathbf{w}_i^\top G \mathbf{w}_i - \lambda(\mathbf{1}^\top \mathbf{w} - \mathbf{1})$

Homework

1. Continuing from previous slide, show that G is invertible.
2. Setting $\partial L / \partial \mathbf{w}_i = \partial L / \partial \lambda = 0$, show that the weights are given by

$$\mathbf{w}_i = \frac{\lambda}{2} G_i^{-1} \mathbf{1}$$

3. Write down the exact value of λ.

LLE - computation

LLE step 3.

- How to find the intrinsic coordinates Y?
- Need to minimize $\Phi(Y) = \sum_{i=1}^{n} \left|\left|\mathbf{y}_i - \sum_{i \neq j} W_{i,j}\mathbf{y}_j\right|\right|^2$
- Let's do the $d = 1$ case. We reduce to one dimension, so that $\mathbf{y}_i$ is actually $y_i \in \mathbb{R}$. Let $\mathbf{y} \in \mathbb{R}^n$ be the vector for all datapoints.
- Define $M = I - W$. Then

$$\begin{aligned}
\Phi(\mathbf{y}) &= \mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top (W\mathbf{y}) - (W\mathbf{y})^\top \mathbf{y} + (W\mathbf{y})^\top (W\mathbf{y}) \\
&= \mathbf{y}^\top (I - W)^\top (I - W)\mathbf{y} \\
&= \mathbf{y}^\top M^\top M\mathbf{y}
\end{aligned}$$

LLE - computation

LLE step 3.

- The constrain $Y^\top Y/n = I$ reduces to $(\mathbf{y}^\top \mathbf{y})/n = 1$.
- Let's add a Lagrange multiplier μ and write the Lagrangian

$$L(\mathbf{y}, \mu) = \mathbf{y}^\top M \mathbf{y} - \mu \left((\mathbf{y}^\top \mathbf{y})/n - 1 \right)$$

- Setting $\partial L / \partial \mathbf{y} = 0$ we obtain

$$M\mathbf{y} = \frac{\mu}{n}\mathbf{y}$$

- **AHA!** The solution is an eigenvector of M!

Homework

1. Show that M has only real non-negative eigenvalues, call them $m_1 \leq \ldots \leq m_n$.
2. Show that M has exactly one zero eigenvalue: $m_1 = 0$ and $m_2 > 0$.
3. In the $d = 1$ case, which eigenvector should be take for the solution **y**?
4. (Difficult.) When $d \geq 1$ (the general case), show that the matrix Y is given by the d eigenvectors corresponding to $m_2, \ldots, m_{d+1}$.

Diffusion Maps

Oops.

- One major drawback of LLE: we don't know what $||\mathbf{y}_i - \mathbf{y}_j||$ mean
- Recall that we set out to find intrinsic coordinates **y**'s where, unlike the **x**'s, the large distances have meaning
- When the data on the manifold, ideally distances between **y**'s should represent distances **on the manifold**
- When data not on manifold (it rarely is), they should be meaningful and we should understand them
- Enter **Diffusion Maps**
- (closely related but not identical to another manifold learning method called **Laplacian Eigenmaps**)

Graph of datapoints

- Let's drop the assumption that the data came to us embedded in Euclidean space, and just assume an abstract dataset
- So imagine we just have n abstract data points, with **distances** or **affinities**.
- In other words, dataset is a **weighted undirected graph**
- Let's work with affinities $K_{i,j} = K_{j,i} \geq 0$. The larger $K_{i,j}$, the closer point i and point j are.
- This is a very useful abstraction
- K is known as a **kernel matrix**
- (Recall the "kernel trick" from IML)

Heat Kernel

- If the data is actually in Euclidean space $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$, one very popular way to build K is using the **heat kernel**: define

$$K_{i,j} = e^{-||\mathbf{x}_i - \mathbf{x}_j||^2/\varepsilon}$$

  where ε is called the **kernel width**

- This makes a lot of sense if indeed $||\mathbf{x}_i - \mathbf{x}_j||$ is meaningless when large (why?)
- Why "heat kernel"? –Related to the Heat PDE

Enter the Random Walk

- Given a kernel matrix K on n abstract datapoints, let's row-normalize it:

$$
\begin{aligned}
D_i &:= \sum_{j=1}^{n} K_{i,j} \\
D &:= diag(D_1, \ldots, D_n) \\
A &:= D^{-1}K
\end{aligned}
$$

- A is a **stochastic matrix** (non-negative entries and normalized rows).
- It is called a **Markov matrix** as it is a transition matrix of a Markov chain on the n abstract datapoints
- This random walk likes to transition between similar datapoints.

Markov chain

- This Markov chain can tell us a lot about the dataset.
- Formally let X_t be the random variable at time $t = 0, 1, 2, \cdots$ (taking values on the dataset, namely on $\{1, \ldots, n\}$. Let's continue to denote the datapoints $x_1, \ldots, x_n$ even though now they're just an abstract set
- Formally,

$$\mathbb{P}(X_{t+1} = x_j \,|\, X_t = x_i) = A_{i,j}$$

Homework

1. Show that the probability of landing at x_k after exactly k chain transitions, if we started from x_i, is

$$\mathbb{P}(X_t = x_k \,|\, X_0 = x_i) = A^t_{i,k}$$

2. Show that $A = D^{-1/2} S D^{1/2}$ where S is symmetric.
3. Conclude that A has orthonormal basis of eigenvectors associated with real eigenvalues.
4. Show that 1 is an eigenvalue. What is the corresponding eigenvector?
5. Let S from above be diagonalized by $S = V \Lambda V^\top$. Show that $A = \Phi \Lambda \Psi^\top$ where $\Phi = D^{-1/2} V$, $\Psi = D^{1/2} V$.

Homework (cont.)

1. Show that all eigenvalues λ of *A* satisfy $|\lambda| \leq 1$ (without using Frobenius-Perron Theorem!...)
2. Suppose that the graph underlying *A* has exactly *r* connected components, what is the multiplicity of the eigenvalue $\lambda = 1$? What are the corresponding eigenvectors?

Diffusion Distance

- Let's define a **meaningful** distance between x_i and x_j. (Recall that when affinity between x_i and x_j is small, it is not meaningful)
- How about the distance between the probability cloud starting from x_i to the probability cloud starting from x_j, after t steps?
- Formally, define the **Diffusion Distance** at time t between x_i and x_j to be

$$\Delta_{i,j}^t = \sqrt{\sum_{k=1}^{n} \frac{1}{d_k} (A_{i,k}^t - A_{j,k}^t)^2}$$

  where $d_k = \sum_j A_{k,j}$.
- When $\Delta_{i,j}^t$ is small, the probability clouds "emanating" from x_i and x_j hardly overlap at time t

Diffusion Maps

- Now consider ϕ_1, ϕ_n, the columns of Φ (right eigenvectors of A)
- Sort them by decreasing order of their eigenvalues, $1 = \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$.
- If the graph underlying A is connected, $\phi_1 = \mathbf{1}$, we should disregard it.
- Suppose we would like to embed the dataset in $\mathbb{R}^{n-1}$ (or, if we started from Euclidean data and use e.g. the heat kernel, reduce dimension to $n-1$)
- Define the Diffusion Map at time t:

$$\Phi_t : x_i \mapsto (\lambda_2^t \phi_2(i), \ldots, \lambda_n^t \phi_n(i))$$

- So that $\Phi_t(x_i) \in \mathbb{R}^{n-1}$

Magic!

- Main Theorem:

$$||\Phi_t(x_i) - \Phi_t(x_j)|| = \Delta^t_{i,j}$$

- So that the Euclidean distances between embedded points **equal** to the diffusion distance between the corresponding points!
- Proof:

$$||\Phi_t(x_i) - \Phi_t(x_j)||^2 = \sum_{k=2}^{n} \lambda_k^{2t} \left(\phi_k(i) - \phi_k(j)\right)^2$$

but also

$$(\Delta^t_{i,j})^2 = \sum_{k=2}^{n} \lambda_k^{2t} \left(\phi_k(i) - \phi_k(j)\right)^2$$

In practice

- We don't use $n-1$ eigenvectors (obviously)
- Instead, choose embedding dimension d as usual and use the map

$$\Phi_t : x_i \mapsto (\lambda_2^t \phi_2(i), \ldots, \lambda_n^t \phi_{d+1}(i))$$

- Since the eigenvalues of A decay very quickly, we have that still $||\Phi_t(x_i) - \Phi_t(x_j)||$ is very close to $\Delta_{i,j}^t$

Multiscale:

- Diffusion maps have the tuning parameter *t* - the diffusion time
- It allows us to control the “resolution” and specify which distances are meaningful

Homework

1. Prove the main theorem above (namely compute both $||\Phi_t(x_i) - \Phi_t(x_j)||^2$ and $(\Delta_{i,j}^t)^2$)

Code

Install Anaconda. Then bash:

```
$ conda create —n APML python=3
$ source activate APML
$ conda install matplotlib
$ conda install seaborn
$ conda install scikit—learn
$ conda install pillow
$ conda install ipykernel
$ jupyter—notebook
```

Coding Homework

1. Use an IPythonNotebook
2. Try both Notebooks shown in class
3. Create a swiss roll and plot it
4. Implement your version of LLE and run it on your swiss roll data
5. Plot the dimension-reduced dataset
6. Experiment with *k*, the number of NN
7. Compare your implementation with that of Python's `scikit-learn`
8. Implement your version of diffusion maps. Kernelize both example datasets with the heat kernel and run your diffusion maps code.
9. Experiment with the diffusion time *t*, the kernel width ε and the embedding dimension *d*
10. Write a tool to visualize both datasets in 3D by using any three eigenvectors ϕ_i, ϕ_j, ϕ_k that the user chooses. Show thumbnails of faces/digits as in examples.

Street fighting

- What to do when n is very large?
- Computational bottlenecks: NN search; diagonalization
- There are fast, approximate methods for both
- (We mentioned fast NN search in passing above)
- Have a look at python package `megaman` (James McQueen et al) which implements some of these fast algorithms and offers most manifold learning algorithms

Credits

Some content adapted from notes by Amit Singer (Princeton), notes by Cosma Shalizi (CMU), Python Data Science Handbook by Jake VanderPlas and `sklearn` docs