CSE 547/STAT 548

Non-linear dimension reduction: an introduction

Marina Meilă

Department of Statistics
University of Washington

January 2022

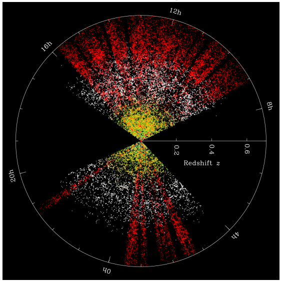
Outline

Who needs manifold learning?

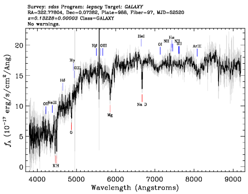
- What is PCA good for?

Spectra of galaxies measured by the Sloan Digital Sky Survey (SDSS)

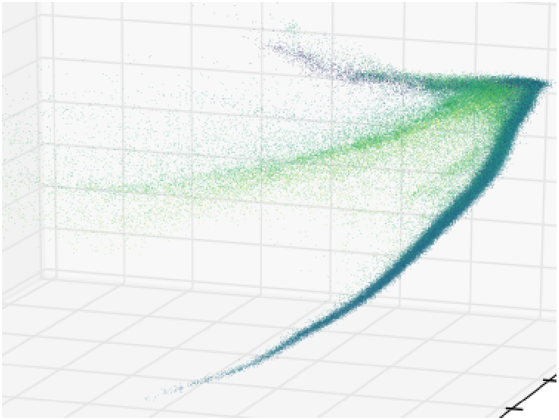
www.sdss.org



www.sdss.org



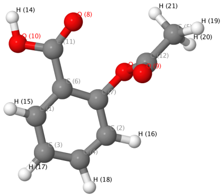
- Preprocessed by Jacob VanderPlas and Grace Telford
- $n = 675,000$ spectra $\times\ D = 3750$ dimensions

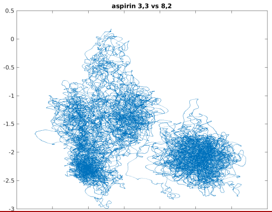


embedding by James McQueen

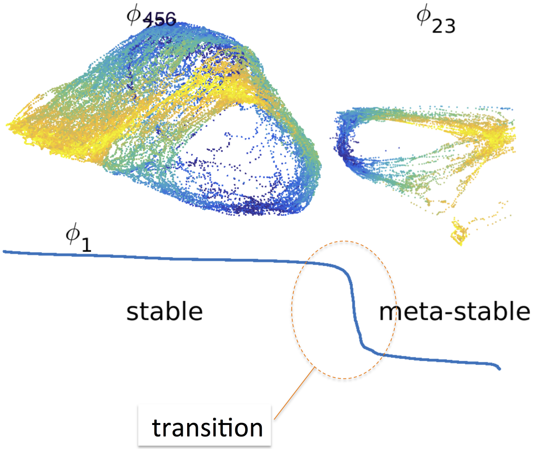
Molecular configurations

aspirin molecule



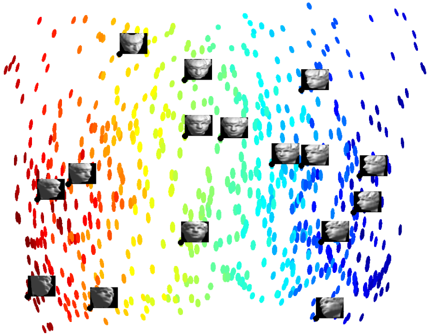


- Data from Molecular Dynamics (MD) simulations of small molecules by [Chmiela et al. 2016]
- $n \approx 200,000$ configurations $\times\, D \sim 20 - 60$ dimensions



When to do (non-linear) dimension reduction

- $n = 698$ gray images of faces in $D = 64 \times 64$ dimensions
- head moves up/down and right/left
- With only two degrees of freedom, the faces define a 2D manifold in the space of all 64×64 gray images



Manifold. Mathematical definitions

Definition 1 (Smooth Manifold (?))

- A d-dimensional manifold $\mathcal{M}$ is a topological (Hausdorff) space such that every point has a neighborhood homeomorphic to an open subset of $\mathbb{R}^d$.
- A *coordinate chart* (U, x) of manifold $\mathcal{M}$ is an open set $U \subset \mathcal{M}$ together with a homeomorphism $x : U \to V$ of U onto an open subset $V \subset \mathbb{R}^d = \{(x^1, ..., x^d) \in \mathbb{R}^d\}$.
- A C^∞-*atlas* $\mathcal{A}$ is a collection of charts, $\mathcal{A} \equiv \cup_{\alpha \in I}\{(U_\alpha, x_\alpha)\}$ where I is an index set, such that $\mathcal{M} = \cup_{\alpha \in I} U_\alpha$ and for any $\alpha, \beta \in I$ the corresponding transition map $x_\beta \circ x_\alpha^{-1} : x_\alpha(U_\alpha \cap U_\beta) \to \mathbb{R}^d$ is continuously differentiable any number of times.

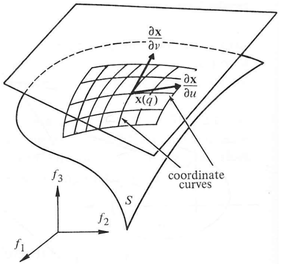
- Notation: $p \in U \;\longrightarrow\; x(p) = (x^1(p), ..., x^d(p))$.
- The mappings $\{x\}$ are not uniquely defined. This is a problem for comparing results of manifold estimation algorithms
- Generally, a manifold needs more than one chart. This is not a severe problem, and can be circumvented as we will see next. For simplicity, we will talk only about a single chart from now on.

Intrinsic dimension. Tangent subspace

- d is called **intrinsic dimension** of $\mathcal{M}$
- If the original data $p \in \mathbb{R}^D$, call D the **ambient dimension**.
- Denote by $\phi : V \subseteq \mathbb{R}^d \to U \subseteq \mathcal{M}$ the inverse of x. A **smooth curve** γ on $\mathcal{M}$ is defined as the image by ϕ of a smooth curve $\tilde{\gamma}$ in V. A smooth curve admits a tangent at every interior point.
- The **tangent subspace** of $\mathcal{M}$ at $p \in \mathcal{M}$, denoted $\mathcal{T}_p\mathcal{M}$ is defined as the set of all tangents at p to smooth curves curves on $\mathcal{M}$ that pass through point p.

$$\dim \mathcal{T}_p\mathcal{M} \; = \; d$$

- If $f : \mathcal{M} \to \mathbb{R}$ is a scalar function on $\mathcal{M}$, then its gradient at p, denoted $\nabla f(p)$, is a vector in $\mathcal{T}_p\mathcal{M}$.
- exterior derivative
- geodesic distance


$\frac{\partial \mathbf{x}}{\partial v}$
$\frac{\partial \mathbf{x}}{\partial u}$
$\mathbf{x}(q)$
coordinate curves
S
f_3
f_2
f_1

Tangents to curves – detail

The Chain Rule $f = h \circ g \Leftrightarrow f(x) = h(g(x))$ where $f : (-1,1) \to U \subset \mathbb{R}^D$, $g : (-1,1) \to V \subset \mathbb{R}^d$, $h : V \to U$

$$\frac{d}{dt} f = dh \frac{d}{dt} g \tag{1}$$

Where $\frac{d}{dt} f \in \mathbb{R}^D$, $\frac{d}{dt} g \in \mathbb{R}^d$, $dh = [\frac{\partial h^i}{\partial x^j}]_{i=1:D}^{j=1:d}$ is the **Jacobian** of h

(Smooth) **Curve** $\bar{\gamma} : (-1,1) \to \mathbb{R}^d$ iff $\bar{\gamma}^j : (-1,1) \to \mathbb{R}$ are smooth functions, for $j = 1 : d$. $\bar{\gamma}(t)$ is point on curve at t.

- Smooth curve on $\mathcal{M}$: $\gamma = \phi \circ \bar{\gamma}$, $\gamma(t) = \phi(\bar{\gamma}^1(t), \ldots \bar{\gamma}^d(t))$
- Hence $\frac{d\gamma}{dt} = d\phi \cdot \frac{d\bar{\gamma}}{dt}$

An example I

- $\mathcal{M}$ is unit sphere in $\mathbb{R}^3$, coordinatex x, y, z
- U is top patch of $\mathcal{M}$. How to map U to $V \subset \mathbb{R}^2$?
  1. We find the inverse mapping $\phi : V \to U$
  2. Let V be a the interior of a circle, coordinates (x^1, x^2) , point $(0,0,1) \in U$ maps to $(0.0) \in V$.
  3. Let $r^2 = (x^1)^2 + (x^2)^2$, and map it to the arc distance from $(0,0,1)$ to $p = (x, y, z)$. Then

$$\begin{aligned} x &= x^1 \sin r \\ y &= x^2 \sin r \\ z &= 1 - \cos r \end{aligned}$$

  4. Let's compute the derivatives (by chain rule)

$$\frac{\partial r}{\partial x^1} = \frac{x^1}{r} \qquad \frac{\partial x}{\partial x^1} = \sin r + \frac{(x^1)^2}{r} \cos r$$

$$\frac{\partial r}{\partial x^2} = \frac{x^2}{r} \qquad \frac{\partial x}{\partial x^2} = \frac{x^1 x^2}{r} \cos r$$

$$\frac{\partial z}{\partial x^1} = \frac{x^1}{r} \sin r \qquad \frac{\partial y}{\partial x^1} = \frac{x^1 x^2}{r} \cos r$$

$$\frac{\partial z}{\partial x^2} = \frac{x^2}{r} \sin r \qquad \frac{\partial y}{\partial x^2} = \sin r + \frac{(x^2)^2}{r} \cos r$$

An example II

- Now let $\bar{\gamma} : (-\epsilon, \epsilon) \to V$ be the curve $\bar{\gamma}(t) = [t\ t]^T$. Hence $\frac{d\bar{\gamma}}{dt} = [1\,1]^T$
- The tangent vector in $p = (0,0,1)$ is $\frac{d\gamma}{dt}(0,0) = d\phi\frac{d\bar{\gamma}}{dt}$ with coordinates

$$\frac{d\gamma}{dt}(0,0) = \left[\begin{array}{l} \sin r + \frac{(x^1)^2 + x^1x^2}{r}\cos r \\ \sin r + \frac{(x^2)^2 + x^1x^2}{r}\cos r \\ \sin r\frac{x^1+x^2}{r} \end{array}\right] \tag{2}$$

Examples of manifolds and coordinate charts

Not manifolds

- dimension not constant
- unions of manifolds that intersect
- sharp corners (non-smooth)
- many/most neural network embeddings
- manifolds can have border

Embeddings

- One can circumvent using multiple charts by mapping the data into $m > d$ dimensions.
- Let $\mathcal{M}$, $\mathcal{N}$ be two manifolds, and $f : \mathcal{M} \to \mathcal{N}$ be a C^∞ (i.e *smooth*) map between them. Then, at each point $p \in \mathcal{M}$, the Jacobian df_p of f at p defines a linear mapping between $T_p\mathcal{M}$, and the tangent subspace to $\mathcal{N}$ at $f(p)$ $T_{f(p)}\mathcal{N}$.

Definition 2 (Rank of a Smooth Map)

A smooth map $f : \mathcal{M} \to \mathcal{N}$ has rank k if the Jacobian $df_p : T_p\mathcal{M} \to T_{f(p)}\mathcal{N}$ of the map has rank k for all points $p \in \mathcal{M}$. Then we write $rank\,(f) = k$.

Definition 3 (Embedding)

Let $\mathcal{M}$ and $\mathcal{N}$ be smooth manifolds and let $f : \mathcal{M} \to \mathcal{N}$ be a smooth injective map, that is $rank(f) = dim(\mathcal{M})$, then f is called an immersion. If $\mathcal{M}$ is homeomorphic to its image under f, then f is an embedding of $\mathcal{M}$ into $\mathcal{N}$.

- Whitney's Embedding Theorem (**?**) states that any d-dimensional smooth manifold can be embedded into $\mathbb{R}^{2d}$.
- Hence, if $d \ll D$, very significant dimension reductions can be achieved with a single map $f : \mathcal{M} \to \mathbb{R}^m$.
- **Manifold learning algorithms** aim to construct maps f like the above from finite data sampled from $\mathcal{M}$.

Non-linear dimension reduction: Three principles

1. Local (weighted) PCA (lPCA)
2. Principal Curves and Surfaces (PCS)
3. Embedding algorithms (Diffusion Maps/Laplacian Eigenmaps, Isomap, LTSA, MVU, Hessian Eigenmaps,. . .)

4. [Other, heuristic] t-SNE, UMAP, LLE

In all cases, given $\mathcal{D} = \{\xi_1, \ldots \xi_m\} \subset \mathcal{M}$, want to "recover" $\mathcal{M}$ of arbitrary shape. What makes the problem hard?

- Intrinsic dimension d
  - must be estimated (we assume we know it)
  - sample complexity is exponential in d – **NONPARAMETRIC**
- non-uniform sampling
- **volume** of $\mathcal{M}$ (we assume volume finite; larger volume requires more samples)
- **injectivity radius/reach** of $\mathcal{M}$
- curvature

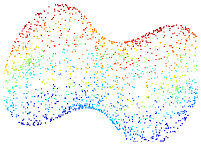
- ESSENTIAL smoothness parameter: the **neighborhood radius** (see next)

Neighborhood graphs

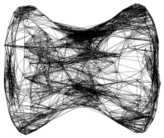
- All ML algorithms start with a **neighborhood graph** over the data points
- In the **radius-neighbor** graph, the neighbors of ξ_i are the points within distance r from ξ_i, i.e. in the ball $B_r(\xi_i)$.
- In the **k-nearest-neighbor (k-nn)** graph, they are the k nearest-neighbors of ξ_i.
- neigh_i denotes the neighbors of ξ_i, and $k_i = |\,\text{neigh}_i\,|$.
- $\Xi_i = [\xi_{i'}]^{i' \in \text{neigh}_i} \in \mathbb{R}^{D \times k_i}$ contains the coordinates of ξ_i's neighbors

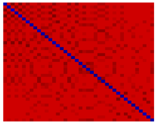
- k-nn graph has many computational advantages
  - constant degree k (or $k-1$)
  - connected for any $k > 1$
  - more software available

  - but much more difficult to use for consistent estimation of manifolds (see later, and)


X

$\xi_1, \ldots \xi_n \subset \mathbb{R}^D$





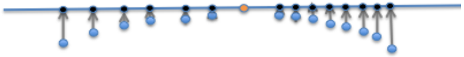
Local PCA

Idea Approximate $\mathcal{M}$ with tangent subspaces at a finite number of data points

1. Pick a point $\xi_i \in \mathcal{D}$
2. Find neigh$_i$, perform PCA on neigh$_i \cup \{\xi_i\}$ and obtain (affine) subspace with basis $T_i \in \mathbb{R}^{D\times d}$
3. Represent $\xi_{i'} \in$ neigh$_i$ by $y_i = \text{Proj}_{T_i}\,\xi_{i'}$

$$y_{i'} = T_i^T(\xi_{i'} - \xi_i) \quad \text{new coordinates of } \xi_{i'} \text{ in } \mathcal{T}_{\xi_i}\mathcal{M} \tag{3}$$

Repeat for a sample of $n' < n$ data points



Local PCA

- For n, n' sufficiently large, $\mathcal{M}$ can be approximated with arbitrary accuracy

- So, are we done? Some issues with lPCA
- Point ξ_j may be represented in multiple T_i's (minor)
- New coordinates y_j are relative to local T_i
- Fine for local operations like regression
- Cumbersome for larger scale operations like following a curve on $\mathcal{M}$

PCA in two ways

Principal Component Analysis

- Data matrix $X = (D \times n)$ each column a data vector
- XX^T is covariance matrix (unnormalized; must be centered!)
- $\text{SVD}(X, d) = U\Sigma V^T$ keep only d principal eigenvectors, and d largest e-values $U = d \times D$ basis vectors
- $Y = U^T X = \Sigma V^T = d \times n$ low dimensional representation of data
- $UU^T X =$ reconstruction of X (D dimensional, rank d)
- Encoding a new $x \in \mathbb{R}^D$: $y = U^T x$

PCA Dual algorithm

- more efficient when $D \gg n$
- Compute $X^T X = K$ Gram matrix (or kernel matrix)
- $\text{EIG}(K, d) = V\Sigma^2 V^T$ keep only d principal eigenvectors, and largest d e-values
- $Y = U^T X = \Sigma V^T = d \times n$ low dimensional representation of data (U not computed unless we want to reconstruct x data)

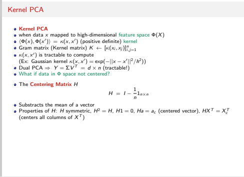
Kernel PCA

- **Kernel PCA**
- when data x mapped to high-dimensional feature space $\Phi(X)$
- $\langle \Phi(x), \Phi(x') \rangle = \kappa(x, x')$ (positive definite) kernel
- Gram matrix (Kernel matrix) $K \leftarrow [\kappa(x_i, x_j)]_{i,j=1}^{n}$
- $\kappa(x, x')$ is tractable to compute
  (Ex: Gaussian kernel $\kappa(x, x') = \exp(-||x - x'||^2/h^2)$)
- Dual PCA $\Rightarrow$ $Y = \Sigma V^T = d \times n$ (tractable!)
- What if data in Φ space not centered?

- The **Centering Matrix** H

$$H = I - \frac{1}{n} 1_{n \times n}$$

- Substracts the mean of a vector
- Properties of H: H symmetric, $H^2 = H$, $H1 = 0$, $Ha = a_c$ (centered vector), $HX^T = X_c^T$ (centers all columns of X^T)



Kernel PCA

- **Kernel PCA**
- when data x mapped to high-dimensional feature space $\Phi(X)$
- $\langle \Phi(x), \Phi(x') \rangle = \kappa(x, x')$ (positive definite) kernel
- Gram matrix (Kernel matrix) $K \leftarrow [\kappa(x_i, x_j)]_{i,j=1}^n$
- $\kappa(x, x')$ is tractable to compute
  (Ex: Gaussian kernel $\kappa(x, x') = \exp(-||x - x'||^2/h^2)$)
- Dual PCA $\Rightarrow$ $Y = \Sigma V^T = d \times n$ (tractable!)
- What if data in Φ space not centered?
- The **Centering Matrix** H

$$H = I - \frac{1}{n} 1_{n \times n}$$

- Substracts the mean of a vector
- Properties of H: H symmetric, $H^2 = H$, $H1 = 0$, $Ha = a_c$ (centered vector), $HX^T = X_c^T$ (centers all columns of X^T)

Exercise 1

Properties of the centering matrix H *Let $a \in \mathbb{R}^n$ be a vector, μ_a the mean of the elements of a,*

$$a_c = a - \mu_a \mathbf{1}_{[\ \ ]} \text{ the centered vector } a. \tag{4}$$

Prove that **a.** *H is symmetric, and* idempotent *$H^2 = H$.*
b. $H1 = 0$
c. $Ha = a_c$
d. *Show that H has an eigenvalue $\sigma_1 = 0$. What is the e-vector for σ_1?*
e. *The eigenvalues of H are $\sigma_1 = 0$, $\sigma_{2:n} = 1$. Characterize the e-vector space for $\sigma_{2:n}$.*
f. *Let $X \in \mathbb{R}^{n \times D}$ a matrix with rows equal to data points in D dimensions. Prove that $X_c = HX$ is a matrix whose rows (as data points) have 0 mean.*
g. *Let $K = XX^T$ be a kernel matrix, and $K_c = X_c X_c^T$. Prove that $K_c = HKH$.*

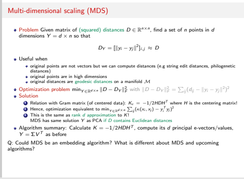
Multi-dimensional scaling (MDS)

- Problem Given matrix of (squared) distances $D \in \mathbb{R}^{n \times n}$, find a set of n points in d dimensions $Y = d \times n$ so that

$$D_Y = [\|y_i - y_j\|^2]_{i,j} \approx D$$

- Useful when
  - original points are not vectors but we can compute distances (e.g string edit distances, philogenetic distances)
  - original points are in high dimensions
  - original distances are geodesic distances on a manifold $\mathcal{M}$
- Optimization problem $\min_{Y \in \mathbb{R}^{d \times n}} \|D - D_Y\|_F^2$ with $\|D - D_Y\|_F^2 = \sum_{ij}(d_{ij} - \|y_i - y_j\|^2)^2$
- Solution
  1. Relation with Gram matrix (of centered data): $K_c = -1/2 HDH^T$ where H is the centering matrix!
  2. Hence, optimization equivalent to $\min_{Y \in \mathbb{R}^{d \times n}} \sum_{ij}(\kappa(x_i, x_j) - y_i^T y_j)^2$
  3. This is the same as rank d approximation to K!
     MDS has same solution Y as PCA if D contains Euclidean distances
- Algorithm summary: Calculate $K = -1/2 HDH^T$, compute its d principal e-vectors/values, $Y = \Sigma V^T$ as before

Q: Could MDS be an embedding algorithm? What is different about MDS and upcoming algorithms?



Multi-dimensional scaling (MDS)

- Problem Given matrix of (squared) distances $D \in \mathbb{R}^{n \times n}$, find a set of n points in d dimensions $Y = d \times n$ so that
$$D_Y = [\|y_i - y_j\|^2]_{i,j} \approx D$$
- Useful when
  - original points are not vectors but we can compute distances (e.g string edit distances, philogenetic distances)
  - original points are in high dimensions
  - original distances are geodesic distances on a manifold $\mathcal{M}$
- Optimization problem $\min_{Y \in \mathbb{R}^{d \times n}} \|D - D_Y\|_F^2$ with $\|D - D_Y\|_F^2 = \sum_{ij}(d_{ij} - \|y_i - y_j\|^2)^2$
- Solution
  1. Relation with Gram matrix (of centered data): $K_c = -1/2HDH^T$ where H is the centering matrix!
  2. Hence, optimization equivalent to $\min_{Y \in \mathbb{R}^{d \times n}} \sum_{ij}(\kappa(x_i, x_j) - y_i^T y_j)^2$
  3. This is the same as rank d approximation to K!
     MDS has same solution Y as PCA if D contains Euclidean distances
- Algorithm summary: Calculate $K = -1/2HDH^T$, compute its d principal e-vectors/values, $Y = \Sigma V^T$ as before

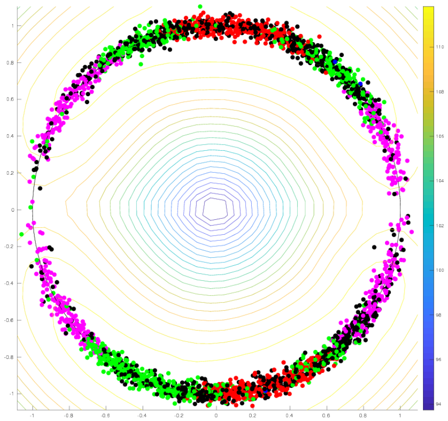
Q: Could MDS be an embedding algorithm? What is different about MDS and upcoming algorithms?

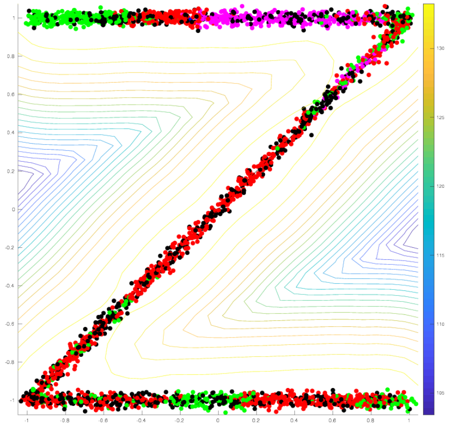
Exercise 2

MDS and Kernel PCA *Prove that* $K_c = -\frac{1}{2} HDH$.

Principal Curves and Surfaces (PCS)

??

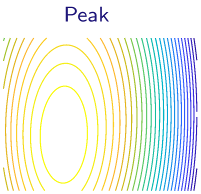




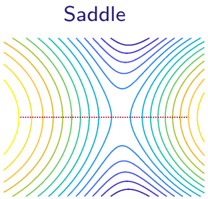
- Elegant algorithm , most useful for $d = 1$ (curves)
- Efficient version by **?**
- Also works in noise **??**

- data in $\mathbb{R}^D$ near a curve (or set of curves)

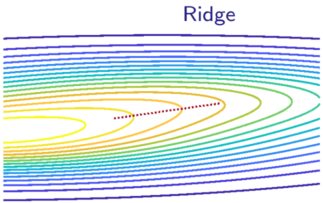
What is a density ridge



$\nabla p = 0$
$\nabla^2 p \prec 0$



$\nabla p = 0$
$\nabla^2 p$ has $\lambda_1 > 0$, $\lambda_{2:D} < 0$



$\nabla p = 0$ in span$\{v_{2:D}\}$
v_j e-vector for λ_j, $j = 1 : D$
$\nabla^2 p$ has $\lambda_{2:D} < 0$

In other words, on a **ridge**

- $\nabla p \propto v_1$ direction of **least negative curvature (LNC)**
- $\nabla p, v_1$ are tangent to the ridge

Gradient and Hessian for Gaussian KDE

- Data $\xi_{1:n} \in \mathbb{R}^D$
- Let p be the kernel density estimator with some kernel width h.

$$p(\xi) \;=\; \frac{1}{nh^d}\sum_{i=1}^{n}\kappa\left(\frac{\xi-\xi_i}{h}\right) \;=\; \frac{1}{nh^d}\sum_{i=1}^{n}\exp\left(-\frac{(\xi-\xi_i)^2}{2h^2}\right)/\omega_d \tag{5}$$

- We prefer to work with $\ln p$ which has the same critical points/ridges as p
- $\nabla \ln p = \frac{1}{p}\nabla p = g$
- $\nabla^2 \ln p = -\frac{1}{p^2}\nabla p \nabla p^T + \frac{1}{p}\nabla^2 p = H$
- $\nabla p(\xi) = \frac{1}{nh^d}\sum_{i=1}^{n} -\underbrace{(\xi-\xi_i)/h^2}_{u_i}\exp\left(-\frac{(\xi-\xi_i)^2}{2h^2}\right)/\omega_d$ hence

$$g(\xi) \;=\; -\frac{1}{h^2}\left[\xi - \sum_{i=1}^{n}\xi_i \underbrace{\exp\left(-\frac{(\xi-\xi_i)^2}{2h^2}\right)/\sum_{i=1}^{n}\exp\left(-\frac{(\xi-\xi_i)^2}{2h^2}\right)}_{w_i}\right] \;=\; -\frac{1}{h^2}\left[\xi - m(\xi)\right] \tag{6}$$

- Mean-shift appears!
- $H(\xi) \;=\; \sum_{i=1}^{n} w_i u_i u_i^T - g(\xi)g(\xi)^T - \frac{1}{h^2}I$

SCMS Algorithm

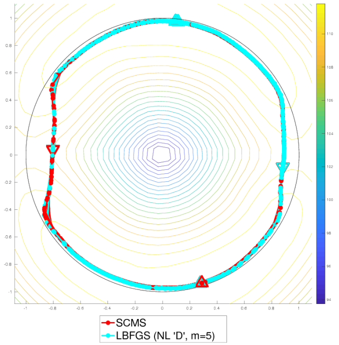
SCMS = Subspace Constrained Mean Shift

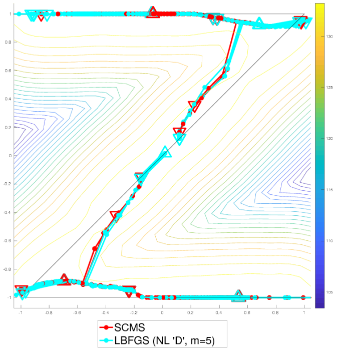
Density estimated by p =data $\star$ Gaussian kernel of width h

```
Init any $x^1$
for $k = 1, 2, \ldots$
    1 calculate $g^k \propto \nabla \ln p(x^k)$                    by Mean-Shift $\mathcal{O}(nD)$
    2 $H^k = \nabla^2 \ln p(x^k)$                                  $\mathcal{O}(nD^2)$
    3 compute $v_1$ principal e-vector of $H^k$                    $\mathcal{O}(D^2)$
    4 $x^{k+1} \leftarrow x^k + \mathrm{Proj}_{v_1^\perp} g^k$      $\mathcal{O}(D)$
until convergence
```

- Algorithm SCMS finds 1 point on ridge; n restarts to cover all density
- Run time $\propto nD^2$/iteration
- Storage $\propto D^2$

Principal curves found by SCMS





LBFGS=accelerated, approximate SCMS – coming next!

Accelerating SCMS

- reduce dependency on n per iteration
  - ignore points far away from ξ
  - use approximate nearest neighbors (clustering, KD-trees,. . .
- reduce number of SCMS runs: start only from $n' < n$ points
- reduce number iterations: **track ridge** instead of cold restarts
  - project ∇p on v_1 instead of $v_1^\perp$
  - tracking ends at critical point (peak or saddle)
- **reduce dependence on D**
  - approximate v_1 without computing whole H
  - $D^2 \leftarrow mD$ with $m \approx 5$

(Approximate) SCMS step without computing Hessian

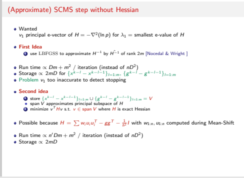
- Given $g \propto \nabla p(x)$
- Wanted $\mathrm{Proj}_{v_1^\perp} g = (I - v_1 v_1^T) g$
- Need v_1

  principal e-vector of $H = \nabla^2(\ln p)$ for $\lambda_1 =$ largest e-value of H

  without computing/storing H

(Approximate) SCMS step without Hessian

- Wanted
  v_1 principal e-vector of $H = -\nabla^2(\ln p)$ for λ_1 = smallest e-value of H
- **First Idea**
  1. use LBFGSS to approximate H^{-1} by $\hat{H^{-1}}$ of rank $2m$ [Nocedal & Wright]
- Run time $\propto Dm + m^2$ / iteration (instead of nD^2)
- Storage $\propto 2mD$ for $\{x^{k-l} - x^{k-l-1}\}_{l=1:m}$, $\{g^{k-l} - g^{k-l-1}\}_{l=1:m}$
- Problem v_1 too inaccurate to detect stopping
- **Second idea**
  1. store $\{x^{k-l} - x^{k-l-1}\}_{l=1:m} \cup \{g^{k-l} - g^{k-l-1}\}_{l=1:m} = V$
     - span V approximates principal subspace of H
  2. minimize $v^T H v$ s.t. $v \in$ span V where H is exact Hessian
- Possible because $H = \sum w_i u_i u_i^T - gg^T - \frac{1}{h^2} I$ with $w_{1:n}, u_{1:n}$ computed during Mean-Shift
- Run time $\propto n' Dm + m^2$ / iteration (instead of nD^2)
- Storage $\propto 2mD$



(Approximate) SCMS step without Hessian

- Wanted
  v_1 principal e-vector of $H = -\nabla^2(\ln p)$ for λ_1 = smallest e-value of H
- **First Idea**
  1. use LBFGSS to approximate H^{-1} by $\hat{H^{-1}}$ of rank $2m$ [Nocedal & Wright]
- Run time $\propto Dm + m^2$ / iteration (instead of nD^2)
- Storage $\propto 2mD$ for $\{x^{k-l} - x^{k-l-1}\}_{l=1:m}, \{g^{k-l} - g^{k-l-1}\}_{l=1:m}$
- Problem v_1 too inaccurate to detect stopping
- **Second idea**
  1. store $\{x^{k-l} - x^{k-l-1}\}_{l=1:m} \cup \{g^{k-l} - g^{k-l-1}\}_{l=1:m} = V$
     - span V approximates principal subspace of H
  2. minimize $v^T H v$ s.t. $v \in \text{span } V$ where H is exact Hessian
- Possible because $H = \sum w_i u_i u_i^T - gg^T - \frac{1}{h^2} I$ with $w_{1:n}, u_{1:n}$ computed during Mean-Shift
- Run time $\propto n' Dm + m^2$ / iteration (instead of nD^2)
- Storage $\propto 2mD$

Exercise 3

Subspace constrained principal e-vector *Let $H \in \mathbb{R}^{D\times D}$ be a symmetric matrix, and $V \in \mathbb{R}^{D\times m}$ an orthogonal matrix defining a subspace. We want to obtain*

$$\underset{v\in \text{span } V}{\operatorname{argmax}} v^T H v \quad \textit{the principal e-vector constrained to } V. \tag{7}$$

a. *Prove that v is the principal e-vector of a symmetric $m \times m$ matrix W. Hint: $v = Vu$ with $u \in \mathbb{R}^m$ for any $v \in$ span V.*
b. *What is W for the Hessian H used in SCMS? and what is the dimension of W in this case?*

Embedding algorithms

- Map $\mathcal{D}$ to $\mathbb{R}^s$ where $s \geq d$ (global coordinates)
- Can also map a local neighborhood $U \subseteq \mathcal{D}$ to $\mathbb{R}^d$ (local, intrinsic coordinates)

Input

- embedding dimension m
- neighborhood radius ϵ
- neighborhood graph, i.e. $\{\text{neigh}_i, \Xi_i, \text{ for } i = 1:n\}$, $A = [\|\xi_i - \xi_j\|]_{i,j=1}^n$ distance matrix $A_{ij} = \infty$ if $i \notin \text{neigh}_j$

The Isomap algorithm

Isomap Algorithm [Tennenbaum, deSilva & Langford 00]

Input A, dimension d

1. Find all shortest path distances in neighborhood graph $A_{ij} \leftarrow$ graph distance between i, j
2. Construct matrix of squared distances

$$M = [(A_{ij})^2]$$

3. use Multi-Dimensional Scaling MDS(M, d) to obtain d dimensional coordinates Y for $\mathcal{D}$

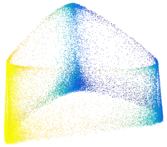
The Diffusion Maps (DM)/ Laplacian Eigenmaps (LE) Algorithm

Diffusion Maps Algorithm

Input distance matrix $A \in \mathbb{R}^{n \times n}$, bandwidth ϵ, embedding dimension s

1. Compute Laplacian $L \in \mathbb{R}^{n \times n}$
2. Compute eigenvectors of L for smallest $s+1$ eigenvalues $[\phi_0\, \phi_1\, \ldots \phi_s] \in \mathbb{R}^{n \times s}$
   - ϕ_0 is constant and not informative
   - These are the slow modes of the system

The **embedding coordinates** of $p_{i:}$ are $(\phi_{i1}, \ldots \phi_{is})$



- **Embedding dimension** s = number of eigenvectors
- **Intrinsic dimension** $d \leq s$ effective number of **degrees of freedeom**

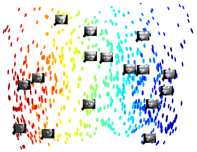
UMAP: Uniform Manifold Approximation and Projection [McInnes, Healy, Melville,2018]

Input k number nearest neighbors, d,

1. Find k-nearest neighbors
2. Construct (asymmetric) similarities w_{ij}, so that $\sum_j w_{ij} = \log_2 k$. $W = [w_{ij}]$.
3. Symmetrize $S = W + W^T - W.*W^T$ is similarity matrix.
4. Initialize embedding ϕ by LaplacianEigenmaps.
5. Optimize embedding.
   Iteratively for n_{iter} steps
   1. Sample an edge ij with probability $\propto \exp -d_{ij}$
   2. Move ϕ_i towards ϕ_j
   3. Sample a random j' uniformly
   4. Move ϕ_i away from $\phi_{j'}$
      Stochastic approximate **logistic regression** of $||\phi_i - \phi_j||$ on d_{ij}.

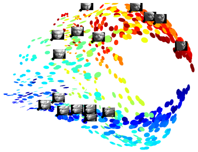
Output ϕ

Isomap vs. Diffusion Maps



Isomap

- Preserves geodesic distances
  - but only when $\mathcal{M}$ is flat and "data" convex
- Computes all-pairs shortest paths $\mathcal{O}(n^3)$
- Stores/processes dense matrix



DiffusionMap

- Distorts geodesic distances
- Computes only distances to nearest neighbors $\mathcal{O}(n^{1+\epsilon})$
- Stores/processes sparse matrix

- t-SNE, UMAP visualization algorithms

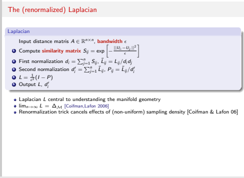
The (renormalized) Laplacian

Laplacian

Input distance matris $A \in \mathbb{R}^{n \times n}$, **bandwidth** ϵ

1. Compute **similarity matrix** $S_{ij} = \exp\left[-\frac{||U_{i:} - U_{j:}||^2}{\epsilon}\right]$
2. First normalization $d_i = \sum_{j=1}^{n} S_{ij}$, $\tilde{L}_{ij} = L_{ij}/d_i d_j$
3. Second normalization $d_i' = \sum_{j=1}^{n} \tilde{L}_{ij}$, $P_{ij} = \tilde{L}_{ij}/d_i'$
4. $L = \frac{1}{\epsilon^2}(I - P)$
5. Output L, d_i'

- Laplacian L central to understanding the manifold geometry
- $\lim_{n\to\infty} L = \Delta_{\mathcal{M}}$ [Coifman,Lafon 2006]
- Renormalization trick cancels effects of (non-uniform) sampling density [Coifman & Lafon 06]



The (renormalized) Laplacian

Laplacian

Input distance matris $A \in \mathbb{R}^{n\times n}$, **bandwidth** ϵ

1. Compute **similarity matrix** $S_{ij} = \exp\left[-\frac{||U_{i:}-U_{j:}||^2}{\epsilon}\right]$
2. First normalization $d_i = \sum_{j=1}^n S_{ij}$, $\tilde{L}_{ij} = L_{ij}/d_i d_j$
3. Second normalization $d_i' = \sum_{j=1}^n \tilde{L}_{ij}$, $P_{ij} = \tilde{L}_{ij}/d_i'$
4. $L = \frac{1}{\epsilon^2}(I-P)$
5. Output L, d_i'

- Laplacian L central to understanding the manifold geometry
- $\lim_{n\to\infty} L = \Delta_{\mathcal{M}}$ [Coifman,Lafon 2006]
- Renormalization trick cancels effects of (non-uniform) sampling density [Coifman & Lafon 06]

Exercise 4

Renormalized Laplacian **a.** *Show that* $L\mathbf{1}_{[]} = 0$ *for the renormalized Laplacian. Hence* L *always has a 0 e-value.*

Exercise 5 (Unnormalized Laplacian)

Let $L^{un} = D - A$ *be the* unnormalized Laplacian *of graph defined by* A*. Prove that* $x^T L^{un} x = \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2$ *for any* $x \in \mathbb{R}^n$*.*

Embedding algorithms summary

- Many different algorithms exist
- All start from neighborhood graph and distance matrix A
- Most use e-vectors of a tranformation of A (preserve the sparsity pattern)

- DiffusionMaps – can separate manifold shape from sampling density
- LTSA – "correct" at boundaries
- Isomap – best for flat manifolds with no holes, small data

- Most embeddings sensitive to
  - choice of radius ϵ (within "correct" range)
  - sampling density p
  - choice of kernel κ, K-nn vs. radius neighbors

    i.e. most embeddings introduce distortions!!

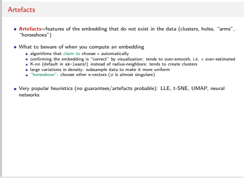
Failures vs. distortions

- Distortion vs failure
  - ϕ distorts if distances, angles, density not preserved, but ϕ smooth and invertible
  - If ϕ does not preserve topology (=preserve neighborhoods), then we call it a failure, for simplicity.
  - Examples: points ξ_i, ξ_j are not neighbors in $\mathcal{M}$ but are neighbors in $\phi(\mathcal{M})$, or viceversa (hence ϕ is not invertible, or not continuous)

- Most common modes of failure
  - A does not capture topology
  - usually becasuse ϵ too small or too large
  - choice of e-vectors

Artefacts

- **Artefacts**=features of the embedding that do not exist in the data (clusters, holes, "arms", "horseshoes")
- What to beware of when you compute an embedding
  - algorithms that claim to choose ϵ automatically
  - confirming the embedding is "correct" by visualization: tends to over-smooth, i.e. ϵ over-estimated
  - K-nn (default in `sk-learn`!) instead of radius-neighbors: tends to create clusters
  - large variations in density: subsample data to make it more uniform
  - "horseshoes": choose other e-vectors (ϕ is almost singulare)
- Very popular heuristics (no guarantees/artefacts probable): LLE, t-SNE, UMAP, neural networks



Artefacts

- **Artefacts**=features of the embedding that do not exist in the data (clusters, holes, “arms”, “horseshoes”)
- What to beware of when you compute an embedding
  - algorithms that claim to choose ϵ automatically
  - confirming the embedding is "correct" by visualization: tends to over-smooth, i.e. ϵ over-estimated
  - K-nn (default in `sk-learn`!) instead of radius-neighbors: tends to create clusters
  - large variations in density: subsample data to make it more uniform
  - "horseshoes": choose other e-vectors (ϕ is almost singulare)
- Very popular heuristics (no guarantees/artefacts probable): LLE, t-SNE, UMAP, neural networks

Exercise 6

Independent coordinates and artefacts for long strips, a,b

a. *Generate a rectangle with a hole. Generate the following sets of points on 2D grids.*

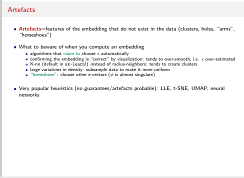
	dimension	*grid spacing*	*number points*
left side	$[0,1]\times[0,1]$	*0.05*	*441*
middle	$[1.01,2]\times[0,0.3]$	*0.01*	$100\times 31 = 3100$
middle	$[1.01,2]\times[0.7,1.]$	*0.01*	$100\times 31 = 3100$
right side	$[2.05,3]\times[0,1]$	*0.05*	*420*
$\mathcal{D}$	$[0,3]\times[0,1]$		*7081*

Plot the data to verify that it is a rectangle with a rectangular hole. The density of the grid is not uniform. In all plots from here on, color the points by their original y coordinate. Ensure that the dot size is small enough for clarity (size 1 or less recommended).

b. *Let $\mathcal{D}$ consist of all the points in* **a.**. *Set the kernel width $\epsilon = 0.05$ and the [optional] neighborhood radius $r = 0.15001$ (i.e. just over 0.15). Calculate for these data*

- *A the distance matrix (can be a dense matrix)*
- *S the similarity matrix (can be a dense matrix)*
- $L^{rw} = I - D^{-1}S$ the random walks *Laplacian*
- *L the renormalized Laplacian*

Display these matrices as square images with an appropriate color scale (don't forget to show the scale with each plot).



Artefacts

- **Artefacts**=features of the embedding that do not exist in the data (clusters, holes, “arms”, “horseshoes”)
- What to beware of when you compute an embedding
  - algorithms that claim to choose ϵ automatically
  - confirming the embedding is "correct" by visualization: tends to over-smooth, i.e. ϵ over-estimated
  - K-nn (default in `sk-learn`!) instead of radius-neighbors: tends to create clusters
  - large variations in density: subsample data to make it more uniform
  - "horseshoes": choose other e-vectors (ϕ is almost singulare)
- Very popular heuristics (no guarantees/artefacts probable): LLE, t-SNE, UMAP, neural networks

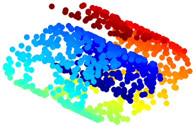
Exercise 7

Independent coordinates and artefacts for long strips - c,d,e,f

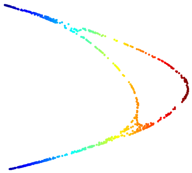
c. *Compute $\phi_{0:9}$ the principal e-vectors $0:9$ for L and discard ϕ_0 the constant vector. Display $\phi_{1:9}$ as a pairwise plot. Ensure that the dot size is small enough for clarity (size 1 or less recommended).*
d. *From the plot in* **c.** *choose a pair of coordinates ϕ_1, ϕ_k that produces the embedding visually closest to the original rectangle. While there is some subjectivity in this choice, embeddings that are “almost dimension 1”, or with self-crossings are NOT close to the original data.*
e. *Repeat* **c,d** *with L^{rw}, denoting its e-vectors $\psi_{0:9}$.*
f. *Embed $\mathcal{D}$ with* Isomap *(OK to use outsourced code) and plot the data in the embedding coordinates y_1, y_2.*

Embedding in 2 dimensions by different manifold learning algorithms

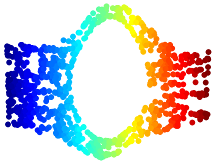
Original data
(Swiss Roll with hole)



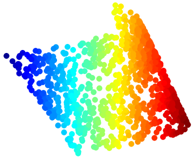
Laplacian Eigenmaps (LE)



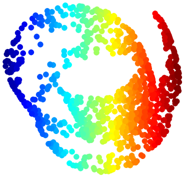
Isomap



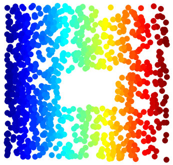
Hessian Eigenmaps (HE)



Local Linear Embedding (LLE)



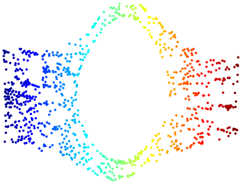
Local Tangent Space Alignment
(LTSA)



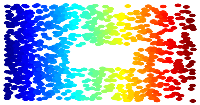
Preserving topology vs. preserving (intrinsic) geometry

- Algorithm maps data $p \in \mathbb{R}^D \longrightarrow \phi(p) = x \in \mathbb{R}^m$
- Mapping $\mathcal{M} \longrightarrow \phi(\mathcal{M})$ is diffeomorphism
  preserves topology
  often satisfied by embedding algorithms
- Mapping ϕ is **isometry**
  - preserves distances along curves in $\mathcal{M}$, angles, volumes
    For most algorithms, in most cases, ϕ is not isometry

Preserves topology



Preserves topology + intrinsic geometry



Theoretical results in isometric embedding

Positive results

General theory

- Nash's Theorem: Isometric embedding is possible.
- Diffusion Maps embedding is isometric in the limit [Berard,Besson,Gallot 94],[Portegies:16]

Special cases

- Isomap [Bernstein, Langford, Tennenbaum 03] recovers flat manifolds isometrically
- LE/DM recover sphere, torus with equal radii (sampled uniformly)
    - Follows from consistency of Laplacian eigenvectors [Hein & al 07,Coifman & Lafon 06, Singer 06, Ting & al 10, Gine & Koltchinskii 06]

Negative results

- Obvious negative examples
- No affine recovery for normalized Laplacian algorithms [Goldberg&al 08]

Empirically, most algorithms

- preserve neighborhoods (=topology)
- distort distances along manifold (=geometry)
- distortions occur even in the simplest cases
- distortion persists when $n \rightarrow \infty$
- one cause of distortion is variations in sampling density p; [Coifman& Lafon 06] introduced Diffusion Maps (DM) to eliminate these

Metric Manifold Learning

Wanted

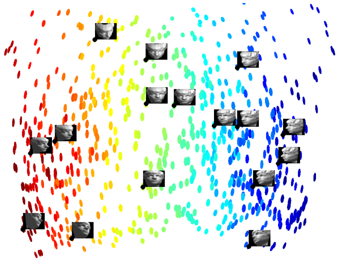
- eliminate distortions for any "well-behaved" $\mathcal{M}$
- and any any "well-behaved" embedding $\phi(\mathcal{M})$
- in a tractable and statistically grounded way

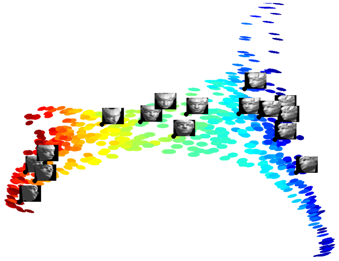
Idea

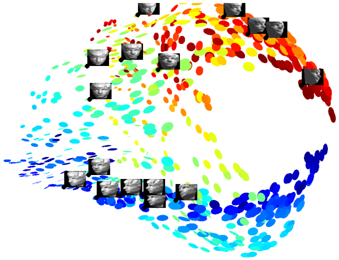
Given data $\mathcal{D} \subset \mathcal{M}$, some embedding $\phi(\mathcal{D})$ that preserves topology (true in many cases)

- Estimate distortion of ϕ and correct it!
- The correction is called the **pushforward Riemannian Metric** g

Corrections for 3 embeddings of the same data


Isomap


LTSA


Laplacian Eigenmaps

Definition 4 (Riemannian Metric)

The Riemannian metric g defines an inner product $<,>_g$ on the tangent space $\mathcal{T}_p\mathcal{M}$ for every $p \in \mathcal{M}$.

Definition 5 (Riemannian Manifold)

A Riemannian manifold $(\mathcal{M}, g)$ is a smooth manifold $\mathcal{M}$ with a Riemannian metric g defined at every point $p \in \mathcal{M}$.

- p point on $\mathcal{M}$
- $\mathcal{T}_p\mathcal{M}$ = **tangent subspace** at p
  at each $p \in \mathcal{M}$, g defines quadratic form G_p

$$< v, w > = v^T G_p w \quad \text{for } v, w \in \mathcal{T}_p\mathcal{M} \text{ and for } p \in \mathcal{M}$$

  - g is symmetric and positive definite tensor field
  - g also called first fundamental form

In coordinates at each point $p \in \mathcal{M}$, G_p is a positive definite matrix of rank d

All (intrinsic) geometric quantities on $\mathcal{M}$ involve g

- Volume element on manifold

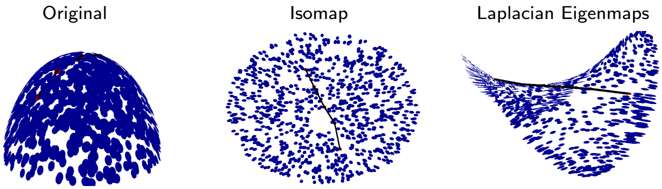
$$Vol(W) = \int_W \sqrt{\det(g)} dx^1 \dots dx^d \,.$$

- Length of curve γ

$$l(\gamma) \;=\; \int_a^b \sqrt{\sum_{ij} g_{ij} \frac{dx^i}{dt} \frac{dx^j}{dt}} dt,$$

- Under a change of parametrization, g changes in a way that leaves geometric quantities invariant

Calculating distances in the manifold $\mathcal{M}$



true distance $d = 1.57$

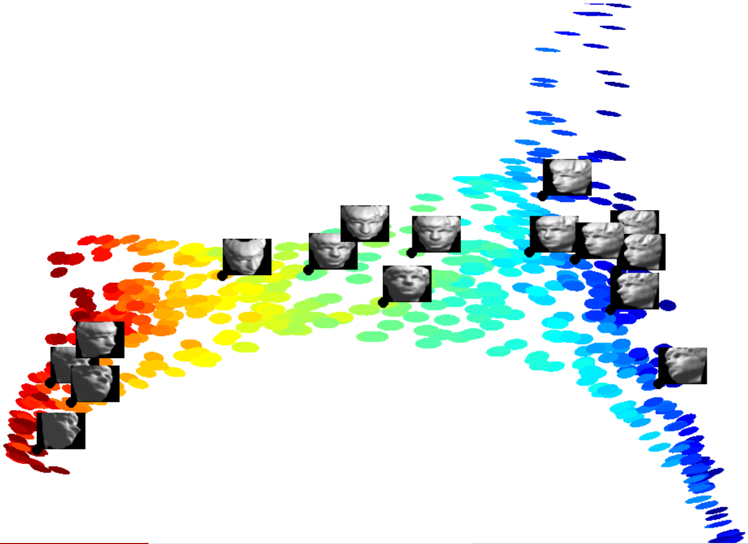
Embedding	$\|\|f(p) - f(p')\|\|$	Shortest Path	Metric $\hat{d}$	Rel. error
Original data	1.41	1.57	1.62	3.0%
Isomap $m = 2$	1.66	1.75	1.63	3.7%
LTSA $m = 2$	0.07	0.08	1.65	4.8%
LE $m = 2$	0.08	0.08	1.62	3.1%

curve $\gamma \approx (y_0, y_1, \ldots y_K)$ path in graph

$$\text{geodesic distance } \hat{d} = \sum_{k=0}^{K} \sqrt{(y_k - y_{k-1})^T G_{ij}(y_k)(y_k - y_{k-1})}$$

G for Sculpture Faces

- $n = 698$ gray images of faces in $D = 64 \times 64$ dimensions
- head moves up/down and right/left



Problem: Estimate the g associated with ϕ

- Given:
  - data set $\mathcal{D} = \{p_1, \ldots p_n\}$ sampled from Riemannian manifold $(\mathcal{M}, g_0)$, $\mathcal{M} \subset \mathbb{R}^D$
  - embedding $\{ y_i = \phi(p_i),\ p_i \in \mathcal{D} \}$
    by e.g DiffusionMap, Isomap, LTSA, ...
- Estimate $G_i \in \mathbb{R}^{m \times m}$ the **pushforward Riemannian metric** at $p_i \in \mathcal{D}$
  in the embedding coordinates ϕ
- The embedding $\{y_{1:n}, G_{1:n}\}$ will preserve the geometry of the original data

Relation between g and Δ

- $\Delta =$ Laplace-Beltrami operator on $\mathcal{M}$
  - $\Delta = \mathrm{div} \cdot \mathrm{grad}$
  - on C^2, $\Delta f = \sum_j \frac{\partial^2 f}{\partial \xi_j^2}$
  - on weighted graph with similarity matrix S, and $t_p = \sum_{pp'} S_{pp'}$, $\Delta = \mathrm{diag}\{ t_p \} - S$
- $\Delta =$ Laplace-Beltrami operator on $\mathcal{M}$
- G Riemannian metric (in coordinates)
- $H = G^{-1}$ matrix inverse

(Differential geometric fact)

$$\Delta f = \sqrt{\det(H)} \sum_l \frac{\partial}{\partial x^l} \left(\frac{1}{\sqrt{\det(H)}} \sum_k H_{lk} \frac{\partial}{\partial x^k} f \right) ,$$

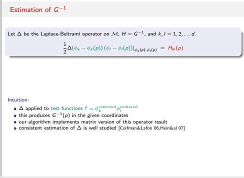
Estimation of G^{-1}

Let Δ be the Laplace-Beltrami operator on $\mathcal{M}$, $H = G^{-1}$, and $k, l = 1, 2, \ldots d$.

$$\frac{1}{2}\Delta(\phi_k - \phi_k(p))\,(\phi_l - \phi_l(p))|_{\phi_k(p),\phi_l(p)} \;=\; H_{kl}(p)$$

Intuition:

- Δ applied to test functions $f = \phi_k^{\text{centered}}\phi_l^{\text{centered}}$
- this produces $G^{-1}(p)$ in the given coordinates
- our algorithm implements matrix version of this operator result
- consistent estimation of Δ is well studied [Coifman&Lafon 06,Hein&al 07]



Estimation of G^{-1}

Let Δ be the Laplace-Beltrami operator on $\mathcal{M}$, $H = G^{-1}$, and $k, l = 1, 2, \ldots d$.

$$\frac{1}{2}\Delta(\phi_k - \phi_k(p))(\phi_l - \phi_l(p))|_{\phi_k(p),\phi_l(p)} = H_{kl}(p)$$

Intuition:

- Δ applied to test functions $f = \phi_k^{\text{centered}}\phi_l^{\text{centered}}$
- this produces $G^{-1}(p)$ in the given coordinates
- our algorithm implements matrix version of this operator result
- consistent estimation of Δ is well studied [Coifman&Lafon 06,Hein&al 07]

this formula includes the change of coordinates. first orderder term s cancels because it’s applied to xi*xj

Metric Manifold Learning algorithm

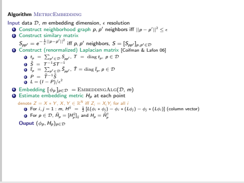
Given dataset $\mathcal{D}$

1. Preprocessing (construct neighborhood graph, ...)
2. Find an embedding ϕ of $\mathcal{D}$ into $\mathbb{R}^m$
3. Estimate discretized Laplace-Beltrami operator L
4. Estimate H_p and $G_p = H_p^\dagger$ for all p
    1. For $i, j = 1 : m$,
       $H^{ij} \;=\; \frac{1}{2}\left[L(\phi_i * \phi_j) - \phi_i * (L\phi_j) - \phi_j * (L\phi_i)\right]$
       where $X * Y$ denotes elementwise product of two vectors $X, Y \in \mathbb{R}^N$
    2. For $p \in \mathcal{D}$, $H_p = [H_p^{ij}]_{ij}$ and $G_p = H_p^\dagger$

Output (ϕ_p, G_p) for all p

Algorithm METRICEMBEDDING

```
Input data $\mathcal{D}$, $m$ embedding dimension, $\epsilon$ resolution
  1 Construct neighborhood graph $p, p'$ neighbors iff $||p - p'||^2 \leq \epsilon$
  2 Construct similary matrix
    $S_{pp'} = e^{-\frac{1}{\epsilon}||p-p'||^2}$ iff $p, p'$ neighbors, $S = [S_{pp'}]_{p,p' \in \mathcal{D}}$
  3 Construct (renormalized) Laplacian matrix [Coifman & Lafon 06]
      1 $t_p = \sum_{p' \in \mathcal{D}} S_{pp'}, \; T = \operatorname{diag} t_p, \; p \in \mathcal{D}$
      2 $\tilde{S} = T^{-1} S T^{-1}$
      3 $\tilde{t}_p = \sum_{p' \in \mathcal{D}} \tilde{S}_{pp'}, \; \tilde{T} = \operatorname{diag} \tilde{t}_p, p \in \mathcal{D}$
      4 $P = \tilde{T}^{-1} \tilde{S}$
      5 $L = (I - P)/\epsilon^2$
  4 Embedding $[\,\phi_p\,]_{p \in \mathcal{D}} = \text{EMBEDDINGALG}(\mathcal{D}, m)$
  5 Estimate embedding metric $H_p$ at each point
    denote $Z = X * Y$, $X, Y \in \mathbb{R}^N$ iff $Z_i = X_i Y_i$ for all $i$
      1 For $i, j = 1 : m$, $H^{ij} = \frac{1}{2}[L(\phi_i * \phi_j) - \phi_i * (L\phi_j) - \phi_j * (L\phi_i)]$ (column vector)
      2 For $p \in \mathcal{D}$, $\tilde{H}_p = [H_p^{ij}]_{ij}$ and $H_p = \tilde{H}_p^{\dagger}$
  Ouput $(\phi_p, H_p)_{p \in \mathcal{D}}$
```



```
Algorithm MetricEmbedding
Input data $\mathcal{D}$, $m$ embedding dimension, $\epsilon$ resolution
  1 Construct neighborhood graph $p, p'$ neighbors iff $||p-p'||^2 \leq \epsilon$
  2 Construct similary matrix
    $S_{pp'} = e^{-\frac{1}{\epsilon}||p-p'||^2}$ iff $p, p'$ neighbors, $S = [S_{pp'}]_{p,p' \in \mathcal{D}}$
  3 Construct (renormalized) Laplacian matrix [Coifman & Lafon 06]
      1 $t_p = \sum_{p' \in \mathcal{D}} S_{pp'}$, $T = \operatorname{diag} t_p$, $p \in \mathcal{D}$
      2 $\tilde{S} = T^{-1} S T^{-1}$
      3 $\tilde{t}_p = \sum_{p' \in \mathcal{D}} \tilde{S}_{pp'}$, $\tilde{T} = \operatorname{diag} \tilde{t}_p$, $p \in \mathcal{D}$
      4 $P = \tilde{T}^{-1} \tilde{S}$
      5 $L = (I - P)/\epsilon^2$
  4 Embedding $[\phi_p]_{p \in \mathcal{D}} = \text{EmbeddingAlg}(\mathcal{D}, m)$
  5 Estimate embedding metric $H_p$ at each point
    denote $Z = X * Y$, $X, Y \in \mathbb{R}^N$ iff $Z_i = X_i Y_i$ for all $i$
      1 For $i, j = 1 : m$, $H^{ij} = \frac{1}{2}[L(\phi_i * \phi_j) - \phi_i * (L\phi_j) - \phi_j * (L\phi_i)]$ (column vector)
      2 For $p \in \mathcal{D}$, $\tilde{H}_p = [H_p^{ij}]_{ij}$ and $H_p = \tilde{H}_p^{\dagger}$
    Ouput $(\phi_p, H_p)_{p \in \mathcal{D}}$
```

This renormalizes the rows of $\tilde{S}$ to sum to 1.

Computational cost

$n = |\mathcal{D}|$, $D =$ data dimension, $m =$ embedding dimension

1. Neighborhood graph $+$
2. Similarity matrix $\quad\mathcal{O}(n^2 D)$ (or less)
3. Laplacian $\mathcal{O}(n^2)$
4. EmbeddingAlg e.g. $\mathcal{O}(mn^2)$ (eigenvector calculations)
5. Embedding metric
    - $\mathcal{O}(nm^2)$ obtain g^{-1} or $h^\dagger$
    - $\mathcal{O}(nm^3)$ obtain g or h

- Steps 1–3 are part of many embedding algorithms
- Steps 3–5 independent of ambient dimension D
- Matrix inversion/pseudoinverse can be performed only when needed

Metric Manifold Learning summary

Why useful

- Measures local distortion induced by any embedding algorithm
  $G_i = I_d$ when no distortion at p_i
- Estimating distortion
- Correcting distortion
  - Integrating with the local volume/length units based on G_i
  - Riemannian Relaxation [McQueen, M, Perrault-Joncas NIPS16]
- Algorithm independent geometry preserving method
- Outputs of different algorithms on the same data are comparable

Applications

- Estimation of neighborhood radius [Perrault-Joncas,M,McQueen NIPS17] and of intrinsic dimension d (variant of [Chen,Little,Maggioni,Rosasco])
- selecting eigencoordinates [Chen, M NeurIPS19]

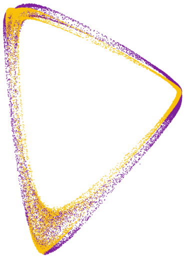
What graph? Radius-neighbors vs. k nearest-neighbors

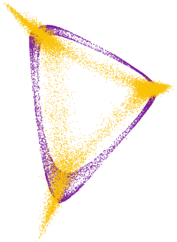
- ***k*-nearest neighbors graph:** each node has degree k
- **radius neighbors graph:** p, p' neighbors iff $||p - p'|| \leq r$

- Does it matter?

- Yes, for estimating the Laplacian and distortion
  - Why? [Hein 07, Coifman 06, Ting 10, ...] k-nearest neighbor Laplacians do not converge to Laplace-Beltrami operator Δ
  - but to $\Delta + 2\nabla(\log p) \cdot \nabla$ (**bias** due to non-uniform sampling)
- Renormalization of Laplacian also necessary

configurations of ethanol $d = 2$


K-nearest neighbor

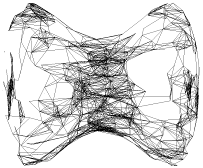
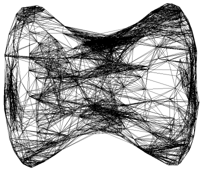
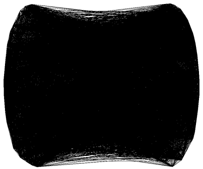

without renormalization

Self-consistent method of chosing ϵ

- Every manifold learning algorithm starts with a neighborhood graph
- Parameter ϵ
  - is neighborhood radius
  - and/or kernel banwidth

- For example, we use the kernel
  $K(p,p') = e^{-\frac{||p-p'||^2}{\epsilon^2}}$ if $||p-p'||^2 \leq \epsilon$ and 0 otherwise

- Problem: how to choose ϵ?

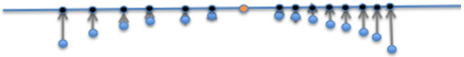




Existing work

- Theoretical (asymptotic) result $\sqrt{\epsilon} \propto n^{-\frac{1}{d+6}}$ [Singer06]
- Visual inspection?
- Cross-validation ?
  - only if related to prediction task
- heuristic for K-nearest neighbor graph [Chen&Buja09]
  - depends on embedding method used
  - K-nearest neighbor graph has different convergence properties than ϵ neighborhood
- Geometric Consistency [Perrault-Joncas&Meila17]
  - Computes "isometry" in 2 different ways and minimizes distortion between them

Geometric Consistency: Idea

- Idea: choose ϵ so that geometry encoded by L_ϵ is closest to data geometry



- For given ϵ and data point p
  1. Project neighbors of p onto tangent subspace
     - this "embedding" is approximately isometric to original data
  2. Calculate Laplacian $L(\epsilon)$) and estimate distortion $H_{\epsilon,p}$ at p
     - $H_{\epsilon,p}$ must be $\approx I_d$ identity matrix

- Completely unsupervised

The distortion measure

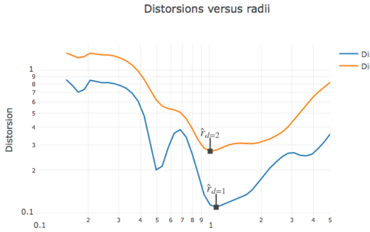
Input: data set $\mathcal{D}$, dimension $d' \leq d$, scale ϵ

1. Estimate Laplacian $L(\epsilon)$ and weights $w_i(\epsilon)$ with LAPLACIAN
2. Project data on tangent plane at p
   - For each p
   - Let $\text{neigh}_{p,\epsilon} = \{p' \in \mathcal{D},\ \|p' - p\| \leq c\epsilon\}$ where $c \in [1, 10]$
   - Calculate (weighted) local PCA (wlPCA) PCA($\text{neigh}_{p,\epsilon}, d'$) (with weights $w_i(\epsilon)$)
   - Calculate coordinates z_i in PCA space for points in $\text{neigh}_{p,\epsilon}$
3. Estimate $H_{\epsilon,p} \in \mathbb{R}^{d' \times d'}$ by RMETRIC
   - For each p
   - Use row p of L
   - z_i's play the role of ϕ
4. Compute quadratic distortion over all p's $D(\epsilon) = \sum_{p\in\mathcal{D}} ||H_{\epsilon,p} - I_d||_2^2$

   Output $D(\epsilon)$

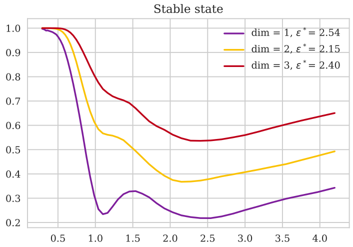
- Select $\epsilon^* = \text{argmin}_\epsilon D(\epsilon)$

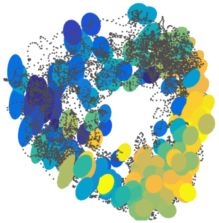
- $d' \leq d$ (more robust)
- H more robust than G
- minimum can be found by 0-th order optimization (faster than grid search)



Example ϵ and distortion for aspirin

- Each point = a configuration of the aspirin molecule
- Cloud of point in $D = 47$ dimensions embedded in $m = 3$ dimensions
- (only 1 cluster shown)

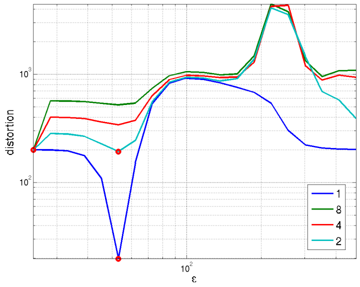




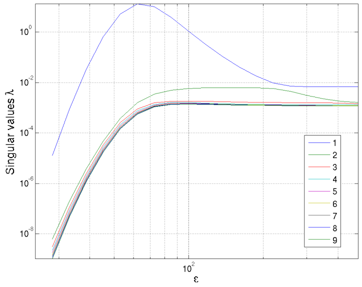
Bonus: Intrinsic Dimension Estimation in noise

- Geometric consistency + eigengap method of [Chen,Little,Maggioni,Rosasco,2011]
  1. do local PCA for a range of neighborhood radii
  2. choose a an appropriate radius ϵ (by Geometric consistency)
  3. dimension = largest eigengap between λ_k and λ_{k+1} at radius ϵ (proof by Chen&al) ("largest" = most frequent largest over a sample)

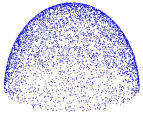
Distortion vs. ϵ

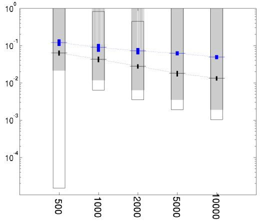


Singular values of lPCA vs. ϵ



Example: Intrinsic Dimension Estimation results




10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
500
1000
2000
5000
10000