The Clustering and Dimension-Reduction *Cookbook*

Anthony Ozerov

2023-05-14

Introduction

What is this? This text is intended as a quick reference on various methods in dimension-reduction and clustering, **not** as an introduction to the subject, or a guide on how to do anything. This text was born out of a lack of a quick reference for clustering and dimension-reduction methods. Of course, there are many texts which discuss some of the methods mentioned here [HTFF09; LLSE11; RB19; MRT18], but we seek to synthesize and condense the information presented further. For each method mentioned, we focus on:

- What is it doing mathematically?
- How do you do it computationally?
- What are some common pitfalls?

We hope that our condensed presentation makes the differences and similarities between different methods clear.

Fair warning Many techniques are omitted either for the sake of brevity or for lack of author knowledge on the subject.

Contribution The `.tex` source for this text is freely available at `https://github.com/anthonyozerov/cdr-cookbook` and licensed under CC0. We welcome contributions.

Acknowledgements We thank Peterson & Pederson's The Matrix Cookbook and Vallentin's Probability and Statistics Cookbook for the inspiration that led us to make this text. We are also grateful to Andrew Blumberg for explaining geometric data analysis so clearly.

Common Notation

Throughout this text, we have sought to strike a balance between having non-overlapping variable names and keeping in line with the literature. Here is a table of the variable names:

Symbol	Meaning
n	The number of points in a dataset
m	The number of points in a subset of a dataset
D	The dimension of the data
d	A dimension $< D$
$d(\cdot)$	A dissimilarity or distance function
$s(\cdot)$	A similarity function
p	The p in p-norm
k	The k in k-NN or k-means (these are different!)
ϵ	The ϵ in ϵ-ball or a small positive number
i and j	Index variables
A and B	Generic sets
X and Y	Metric spaces / sets of data points
P and Q	Probability measures on a metric sapce
C	A collection of sets (e.g. a clustering)

Contents

1 Similarity, Dissimilarity, and Distance

1.1 Definitions

For the rest of this text, we will be making use of ideas of similarity, dissimilarity, and distance. We follow the convention of [HTFF09] in our terminology.

A measure of **similarity** is a function denoting how similar two mathematical objects are. 0 indicates the least possible similarity, and large numbers indicate high similarity.

A measure of **dissimilarity** does the opposite of a measure of dissimilarity, instead denoting how not-similar two mathematical objects are. 0 indicates the most possible similarity (in some cases, this will only occur when the two objects are equal), and large numbers indicate low similarity.

Distance, as we will use it throughout this text, is a refined notion of dissimilarity referred to as a **distance metric**. A function $d : X \times X \to \mathbb{R}$ must satisfy the following properties to be a distance metric:

1. $\forall x \in X \; d(x, x) = 0$
2. $\forall x \neq y \in X \; d(x, y) > 0$ (Positivity/Separation)
3. $\forall x, y \in X \; d(x, y) = d(y, x)$ (Symmetry)
4. $\forall x, y, z \in X \; d(x, y) \leq d(x, y) + d(y, z)$ (Triangle Inequality)

A tuple (X, d) of a set X and a distance metric on X is a **metric space**. Sometimes a function doesn't seem to be a distance metric until X is very carefully specified. A function that satisfies all properties but positivity is a **pseudometric**. A function that satisfies all properties but symmetry is a **quasi-metric**. The word **semimetric** is not used consistently in the literature, but often means a function that satisfies all properties but the triangle inequality.

Throughout this section, similarity measures will be labeled `S`, dissimilarity mesures will be labeled `D`, distance metrics will be labeled `M`, and pseudometrics will be labeled `P`. We will name some of the key dissimilarity functions, and leave others as a generic d.

1.2 Similarities to dissimilarities and back

Sometimes, we may only have a measure of dissimilarity, but need a similarity, or vice-versa. In principle, for any strictly decreasing function $f : [0, \infty] \to [0, \infty]$, we can let similarity be f(dissimilarity) or let dissimilarity be f(similarity). Here are some options for converting a similarity s to a dissimilarity d [MGL22]:

$$
\begin{aligned}
d(x, y) &= 1/s(x, y) \\
d(x, y) &= \min_{x', y'} s(x', y') + \max_{x', y'} s(x', y') - s(x, y) \\
d(x, y) &= -\log\left(s(x, y) / \max_{x', y'} s(x', y')\right)
\end{aligned}
$$

Here are some options for converting a dissimilarity d to a similarity s [EMTB00]:

$$s(x,y) = \max_{x',y'} d(x',y') - d(x,y)$$
$$s(x,y) = \sqrt{\max_{x',y'} d(x',y') - d(x,y)}$$
$$s(x,y) = \max_{x',y'} d(x',y')^2 - d(x,y)^2$$
$$s(x,y) = 1 - \frac{d(x,y)}{\max_{x',y'} d(x',y')}$$
$$s(x,y) = \frac{1}{1+d(x,y)}$$

1.3 Normed vector spaces

In a vector space where a norm is defined (e.g. Euclidean space), there are many induced dissimilarity measures that we can compute.

1.3.1 Minkowski distance [D/M]

The Minkowski distance of order p is:

$$d_p(x,y) = ||x-y||_p = \left(\sum_{i=1}^{D} |x_i - y_i|^p\right)^{1/p}$$

We go through some specific settings of p and some limiting cases below.

1.3.2 Minkowsky as $p \to -\infty$ [D]

$$d(x,y) = \lim_{p\to-\infty} ||x-y||_p = \min_i(|x_i - y_i|)$$

To our knowledge there is no name for this in the literature.

1.3.3 Minkowsky as $p \to 0^+$ [M]

$$d(x,y) = \lim_{p\to 0^+} \left(\sum_{i=1}^{D} |x_i - y_i|^p\right)^{1/p} = \mathbf{1}(x \neq y) = \begin{cases} 1 & x \neq y \\ 0 & x = y \end{cases}$$

1.3.4 Manhattan Distance [M]

This is Minkowski with $p = 1$. If you're walking in a city with a grid system, this is the distance from point x to point y:

$$d(x,y) = ||x-y||_1 = \sum_{i=1}^{d} |x_i - y_i|$$

1.3.5 Euclidean Distance [M]

This is Minkowski with $p = 2$:

$$d_{\mathbb{R}^D}(x,y) = ||x-y||_2 = \sqrt{\sum_{i=1}^{D} |x_i - y_i|^2}$$

1.3.6 Chebyshev distance [M]

This is Minkowski as $p \to \infty$:

$$d(x, y) = \lim_{p\to\infty} ||x - y||_p = \max_i(|x_i - y_i|)$$

1.4 Inner product spaces

1.4.1 Cosine Similarity [S]

$$d(x, y) = \frac{x \cdot y}{||x||\ ||y||}$$

1.5 Curved spaces

1.5.1 Great-circle distance on a 2-sphere [M]

In geospatial data, we often have data points on a sphere like the Earth, expressed in latitude and longitude. If the points are spread over too large portion of the sphere's surface for distances in a local Euclidean tangent plane to be useful (or we are too lazy to project to a plane), then we can use the haversine formula. Suppose x has latitude and longitude ϕ_1, λ_1, y has latitude and longitude ϕ_2, λ_2, and the radius of the sphere is r. Then: [Gad10]

$$d(x, y) = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Considerations On the Earth, which is not a perfect sphere, this is only an approximation. If using a computer, beware of numerical instabilities if two points are antipodal.

1.6 Spaces of sets

Let X and Y be two metric spaces. Let A and B be subsets of X.

1.6.1 Hausdorff [P/M]

$$d_H(A, B) = \inf\{\epsilon : A \subseteq B_\epsilon, B \subseteq A_\epsilon\}$$

Considerations Because, if A and B have the same closure, $d_H(A, B) = 0$, the Hausdorff distance is a metric only if we define it over equivalence classes of sets which have the same closure, or only consider closed subsets of our space.

1.6.2 Gromov-Hausdorff [P/M]

If X and Y are different metric spaces, or they are in the same metric space but we want to measure distance up to rigid transformations, then we can use the Gromov-Hausdorff distance:

$$d_{GH}(X, Y) = \inf_{\substack{\theta_1: X\to Z \\ \theta_2: Y\to Z \\ \theta_i \text{ isometry}}} d_H(\theta_1(X), \theta_2(Y)).$$

Computation This is difficult to compute in practice, but if X and Y are finite:

$$d_{GH}(X, Y) = \tfrac{1}{2} \inf_{R\subseteq X\times Y} \sup_{\substack{(x,y)\in R \\ (x',y')\in R}} |d_X(x, x') - d_Y(y, y')|,$$

which is more tractable. A useful property is that $d_{GH}(X, Y) \leq \frac{1}{2}\text{diam}(X)\text{diam(Y)}$.

Considerations Similarly to the Hausdorff distance, if A and B are different but isometric, $d_{GH}(A, B) = 0$. So the Gromov-Hausdorff distance is a metric only if we define it over equivalence classes of isometric metric spaces.

1.7 Spaces of probability distributions

1.7.1 Total Variation Distance [M]

Let P and Q be two probability measures on a metric space X.

$$d_{TV}(P, Q) = \sup_{x \in X} |P(x) - Q(x)|$$

As a useful inequality, we have that:

$$d_{TV}(P, Q) \leq \sqrt{d_{KL}(P, Q)/2} \qquad \text{(Pinsker's Inequality [Tsy08])}$$

1.7.2 Wasserstein/Earth-Mover/Optimal-Transport/Kantorovich Distance [M]

We follow [Was17]. Let P be a probability measure on X, and Q be a probability measure on Y. Let $\mathscr{J}$ denote the set of all joint distributions over (X, Y) that have marginals P and Q.

$$d_{W_p}(P, Q) = \left(\inf_{J \in \mathscr{J}(P,Q)} \int_{X \times Y} d(x, y)^p dJ(x, y) \right)^{1/p}$$

Computation Computing this is easy in 1-dimensional space, as there the J which realizes the infimum is just the joint distribution which maps quantiles of P to quantiles of Q.

1.7.3 Kullback-Leibler (KL) Divergence / Relative Entropy [D]

Let P and Q be probability measures on X. If X is discrete:

$$d_{KL}(P, Q) = \sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

If X is not discrete:

$$d_{KL}(P, Q) = \int_X P(x) \log \frac{P(x)}{Q(x)} dx$$

If P_1 and Q_1 are probability measures on X, P_2 and Q_2 are probability measures on Y, and $P = P_1 P_2$ and $Q = Q_1 Q_2$, then we have that:

$$d_{KL}(P, Q) = d_{KL}(P_1, Q_1) + d_{KL}(P_2, Q_2) \qquad \text{[Tsy08]}$$

Considerations KL Divergence doesn't satisfy the triangle inequality. And unlike most measures of dissimilarity, it doesn't satisfy symmetry. We can, however, symmetrize by taking $D_{KL}(P, Q) + D_{KL}(Q, P)$, obtaining what is sometimes called "KL Distance," though it is also not a metric. [IM07; Tsy08].

1.8 Spaces of text or strings

Sometimes our data can be thought of as a sequence of symbols, such as "ahhhhhh" and "cat." (see for example, genetic data). How can we define a distance between such objects? We follow [Nav01] in our discussion.

First let's define four operations:

- Insertion. We insert one character somewhere into the string.
- Deletion. We delete one character somewhere in the string.
- Substitution. We replace one character in the string with some character.
- Transposition. We swap two adjacent characters in the string.

1.8.1 Levenshtein/edit distance [M]

This is the minimum number of one-character insertions, deletions, and substitutions it takes to get from string x to string y or vice-versa.

Computation Luckily there is a simple recursive formula:

$$d(x,y) = \begin{cases} |x| & |y| = 0 \\ |y| & |x| = 0 \\ d(x_{2,\dots,|x|}, y_{2,\dots,|y|}) & x_1 = y_1 \\ 1 + \min \begin{cases} d(x_{2,\dots,|x|}, y) \\ (x, y_{2,\dots,|y|}) \\ d(x_{2,\dots,|x|}, y_{2,\dots,|y|}) \end{cases} & \text{o.w.} \end{cases}$$

1.8.2 Hamming distance [M]

This is the minimum number of substitutions to get from string x to string y or vice-versa.

Computation This is just the number of characters which differ between the two strings.

Considerations The two strings must be of equal length, as otherwise we cannot turn one into the other only with substitutions.

1.8.3 Episode distance [D]

This is the minimum number of insertions to get from one string to the other.

Considerations This is not symmetric. If $|x| > |y|$, then $d(x,y) = \infty$. Similarly if y does not contain x as a (not necessarily consecutive) subsequence, then $d(x,y) = \infty$. We see that actually $d(x,y)$ is either ∞ or $|y| - |x|$.

1.8.4 Longest Common Subsequence [M]

This is the minimum number of insertions and deletions to get from string x to string y.

Computation This is the length of the longest common (ordered) subsequence. This is a well-studied problem in algorithms and admits a recursive solution. Let $LCS(i, j)$ denote the length of the longest common subsequence between $x_{1,\dots,i}$ and $y_{1,\dots,j}$, such that $d(x, y) = LCS(|x|, |y|)$. Then we can make the following recurrence relation: [CLRS22]

$$LCS(i,j) = \begin{cases} 0 & i = 0 \text{or} x = 0 \\ LCS(i-1, j-1) + 1 & i, j > 0, x_i = y_j \\ \max(LCS(i, j-1), LCS(i-1, j) & i, j > 0, x_i \neq y_j \end{cases}$$

1.9 Building graphs

For any of the graphs below, if we want them to be weighted we can give edge (x, y) the weight $d(x, y)$.

1.9.1 k-nn graph

In this graph the vertices are the points of the dataset and the edges are between nearest neighbors:

$$G = (V, E) : \ V = X, \ E = \{(x, y) | y \in k\text{-NN of } x\}$$

This graph can be treated as directed or undirected.

Considerations If treated as directed, it is possible that there is an edge (x, y) and no edge (y, x), which may or may not be desirable.

1.9.2 ϵ-ball graph

In this graph the vertices are the points of the dataset and the edges are between points within a distance of ϵ of each other:

$$G = (V, E) : \ V = X, \ E = \{(x, y) | d(x, y) < \epsilon\}$$

If d is symmetric, this graph will behave the same whether directed or undirected. In principle though a non-symmetric dissimilarity measure d could be used.

Computation This graph is easy to compute with matrix operations as the adjacency matrix will simply be 1 where the distance matrix is $\leq \epsilon$, and 0 everywhere else.

Considerations It is very easy in practice to make a disconnected graph if a small ϵ is chosen, which may or may not be desirable for downstream tasks.

1.10 Spaces on graphs

Using the graphs in the subsection above, we can compute certain distances and similarities.

1.10.1 Path metric `[M]`

$$d(x, y) = \{\text{Path distance from } x \text{ to } y \text{ in } G\}$$

In practice this can be computed efficiently by e.g. Dijkstra's algorithm.

1.10.2 Adjacency matrix [S]

This is a measure of similarity which is simply the adjacency matrix of the graph:

$$s(x,y) = \begin{cases} 1; & (x,y) \in E \\ 0; & (x,y) \notin E \end{cases}$$

1.10.3 Heat Kernel [S]

This is a measure of similarity which is a bit more involved than simply the adjacency matrix:

$$s(x,y) = \begin{cases} \exp\left(\frac{-d(x,y)^2}{2\sigma^2}\right); & (x,y) \in E \\ 0; & (x,y) \notin E \end{cases}$$

Considerations The choice of σ will affect downstream tasks. Decreasing σ increases the difference in the similarities between close and distant points.

2 Dimension-reduction

2.1 Definitions

2.1.1 Useful Matrices

Having defined these variables, we can now define a several useful matrices used in dimension-reduction. Let n be the number of data points in our dataset, D be the dimension of our data, and d be the target dimension.

Symbol	___ matrix	Calculation	Dimension	PSD
$\mathbf{X}$	Data	$X_{i*} = x_i$	$n \times D$	No
$\mathbf{D}$	Distance	$D_{ij} = d(x_i, x_j)$	$n \times n$	Often
$\mathbf{D}^{(2)}$	Squared distance	$D^{(2)}_{ij} = (D_{ij})^2$	$n \times n$	Often
$\mathbf{W}$	Weight/similarity/adjacency	*Various*	$n \times n$	No
$\mathbf{D_g}$	Degree	$\operatorname{diag}\left(\sum_j W_{ji}\right)$	$n \times n$	Yes
$\mathbf{L}$	Laplacian	$\mathbf{D_g} - \mathbf{W}$	$n \times n$	Yes
$\mathbf{G}$	Gram	$G_{ij} = \langle x_i, x_j \rangle$	$n \times n$	Yes
$\mathbf{I}_n$	Identity	$I_{n,ij} = \mathbf{1}_{[i=j]}$	$n \times n$	Yes
$\mathbf{C}_n$	Centering	$\mathbf{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^t$	$n \times n$	Yes
$\tilde{\mathbf{X}}$	Centered data	$\mathbf{C}_n\mathbf{X}$	$n \times D$	No
$\mathbf{C}$	Empirical covariance	$\frac{\tilde{\mathbf{X}}\,\tilde{\mathbf{X}}^T}{n}$ or $\frac{\tilde{\mathbf{X}}\,\tilde{\mathbf{X}}^T}{n-1}$	$D \times D$	Yes
$\mathbf{R}$	Random	*Various*	$D \times d$	No
$\mathbf{P}$	Projection	*Various*	$D \times D$	No
$\mathbf{K}$	Kernel	$K_{ij} = K(x_i, x_j)$	$n \times n$	Often

Note that the data matrix is defined as $n \times D$ (so rows are observations, columns are features), and not $D \times n$. The convention varies in different fields and subfields, but we will consistently use $n \times D$, so some of the equations below may be transposes of versions common in the literature.

2.1.2 Top and bottom eigenvectors

The **bottom-k eigenvectors** of a matrix are the eigenvectors corresponding to the matrix's k-smallest eigenvalues. The **top-k eigenvectors** are the eigenvectors corresponding to the k-largest eigenvalues.

2.2 Principal Component Analysis

2.2.1 Linear PCA

Definition Given p-dimensional data $x_1, \ldots, x_n$, we want to choose a d-dimensional subspace of $\mathbb{R}^D$ where, once we project each point to this subspace to obtain $y_1, \ldots, y_n$. If we consider $y_1, \ldots, y_n$ as still being points in $\mathbb{R}^d$, we minimize:

$$\sum_{i=1}^{n} d_{\mathbb{R}^D}(x_i, y_i)^2 = ||\mathbf{XP} - \mathbf{X}||_F^2,$$

where $\mathbf{P}$ is a $D \times D$ rank-d orthogonal projection matrix. The norm on the right is the Frobenius norm.

Computation

1. Compute the covariance matrix $\mathbf{C}$ of the data.
2. Let $\mathbf{V}$ be the $D \times d$ matrix of the bottom d eigenvectors of $\mathbf{C}$.
3. Our embedded data is $\mathbf{Y} = \mathbf{XV}$

Considerations Since the map is linear, if we think our data come from a manifold that is not a hyperplane, PCA will perform poorly.

2.2.2 Kernel/nonlinear PCA

Linear PCA obtains a linear map $\mathbf{V}$ which multiply by our data $\mathbf{X}$ to obtain $\mathbf{Y}$

Rather than just projecting $\mathbf{X}$, let's suppose we have some nonlinear function $\phi : \mathbb{R}^D \to \mathbb{R}^{D'}$, where $D' \gg D$. Denote by $\mathbf{X}_\phi$ the data matrix when ϕ is applied to the data. We are now interested in minimizing:

$$||\mathbf{X}_\phi \mathbf{P} - \mathbf{X}_\phi||_F^2$$

Where $\mathbf{P}$ is now a $D' \times d$ projection matrix. We are essentially looking for projections of a higher-dimensional representation of X.

It turns out that we can *kernelize* the method and never have to actually compute $\mathbf{X}_\phi$. This requires that we have a **kernel function** $K : X \times X \mapsto \mathscr{R}$ where $K(x, y) = \langle \phi(x), \phi(y) \rangle$. In fact, we can completely ignore ϕ and the inner product and just imagine that we have some function $K : X \times X \mapsto \mathscr{R}$ which, when represented as a matrix, is positive semi-definite.

Computation To do Kernel PCA we can use the following approach [MRT18]:

1. Compute the $n \times n$ kernel matrix $\mathbf{K}$ according to $K_{ij} = K(x_i, x_j)$.
2. Let $\mathbf{V}$ be the $n \times d$ matrix of the bottom d eigenvectors of $\mathbf{K}$.
3. If we let row i of $\mathbf{V}$ be $\mathbf{v}_i$, then our embedding for x_i is $\mathbf{v}_i$ elementwise multiplied with the vector $(\lambda_1, \ldots, \lambda_d)$ of eigenvalues of $\mathbf{K}$.

Several other dimension-reduction methods can be represented as Kernel PCA with a specific $\mathbf{K}$. Where this is possible, we will note this below.

2.3 Random Projections

We follow [LHC06] in our discussion of different projection matrices.

2.3.1 General method

Definition Given data $\mathbf{X}$, we create a random matrix $D \times d$ matrix $\mathbf{R}$ with entries $\{R_{ij}\}$ which are iid with mean zero, and let our projected data be $\mathbf{Y} = \frac{1}{\sqrt{d}}\mathbf{XR}$.

2.3.2 Johnson-Lindenstrauss lemma

For some $0 < \epsilon < 1$, let $d \in \mathbb{N}$ be such that:

$$d \geq 24 \ln n/(3\epsilon^2 - 2\epsilon^3) \text{ [DG03]}$$
$$d \geq 20 \ln n/\epsilon^2;\ n > 4 \quad \text{[MRT18]}$$

Then $\exists\ f : \mathbb{R}^D \to \mathbb{R}^d$ with:

$$(1-\epsilon)d_{\mathbb{R}^D}(x,y)^2 \leq d_{\mathbb{R}^d}(f(x),f(y))^2 \leq (1+\epsilon)d_{\mathbb{R}^D}(x,y)^2$$

This lemma does not on its own justify random projections (as a satisfying f might be nonlinear). However, one way to prove the lemma is a probabilistic argument which shows that, if we take a projection to a random d-dimensional subspace of $\mathbb{R}^D$, and let f be that projection scaled by $\sqrt{n/d}$, the probability that f doesn't satisfy the error bounds is $\leq 1 - \frac{1}{n} < 1$. Since there is a pretty decent probability $\geq 1/n$ of a random projection satisfying the bounds, random projections are a good idea.

2.3.3 Conventional random projections

$$R_{ij} \sim N(0,1)$$

Considerations If the data is very-high dimensional, computing $\frac{1}{\sqrt{d}}\mathbf{XR}$ will be computationally expensive, as we can't use any sparsity in $\mathbf{R}$.

2.3.4 Sparse random projections

$$R_{ij} = \sqrt{s}\begin{cases} 1 & \text{w.p. } 1/2s \\ 0 & \text{w.p. } 1 - 1/s \\ -1 & \text{w.p. } 1/2s \end{cases}$$

Computation If the multiplication by $\sqrt{s}$ is done after projecting, then the projection can be done via a bunch of highly efficient database aggregations because only multiplication by 1, 0, and -1 is required.

2.3.5 Very sparse random projections

Sparse random projections (2.3.4) with $s = o(D)$, e.g. $s = \sqrt{D}$ or $s = D/\log D$. See [LHC06] for discussion of the error bounds.

2.4 Multidimensional Scaling (MDS)

We follow [BG05] in our discussion.

2.4.1 Metric MDS

In metric MDS, for some p and some f mapping our points $\mathbf{X}$ to a d-dimensional Euclidean space, we want to minimize the stress function:

$$\left(\sum_{i\neq j}[d(x_i,x_j)^p - d_{\mathbb{R}^d}(f(x_i),f(x_j))]^2\right)^{1/2}$$

2.4.2 Classical MDS

Definition In Classical MDS, we set $p=1$ in the above equation. The strain function which we want to minimize can then be written as:

$$\left(\frac{\sum_{i,j}((\mathbf{B})_{ij} - f(x_i)^T f(x_j))^2}{\sum_{i,j}(\mathbf{B})_{ij}^2}\right)^{1/2},$$

where $\mathbf{B}$ is as defined below. When the distances are Euclidean, classical MDS is equivalent to PCA.

Computation Given distance matrix $\mathbf{D}$ and number of points n, if we want to reduce to d dimensions, this problem can be solved using the following algorithm:[Wic03]

1. Given a distance matrix $\mathbf{D}$, compute the elementwise squared distance matrix $\mathbf{D}^{(2)}$.
2. Let $\mathbf{B} = -\frac{1}{2}\mathbf{C}_n\mathbf{D}^{(2)}\mathbf{C}_n$
3. Compute the d largest positive eigenvalues $\lambda_1,\ldots,\lambda_d$ of $\mathbf{B}$ and their corresponding eigenvectors $e_1,\ldots,e_d$
4. Let $\mathbf{V}$ be the $n\times d$ matrix of eigenvectors and $\mathbf{\Lambda}$ be the $d\times d$ diagonal matrix of $\lambda_1,\ldots,\lambda_d$. The new $n\times d$ embedding of our data is $\mathbf{Y} = \mathbf{V}\mathbf{\Lambda}^{1/2}$.

If n is very large, doing MDS on the whole dataset may be expensive. Instead, we can choose a small subset of $m \ll n$ "landmark" points, embed them in the lower-dimensional space, then use the distances of the rest of the points to the landmark points to embed them in the lower-dimensional space too. Here is an outline of how this, called **Landmark MDS** is done [DT04]:

1. Choose a subset of $m \ll n$ points and compute their $m\times m$ distance matrix $\mathbf{D}$.
2. Do MDS as above using this distance matrix to obtain an embedding $\mathbf{Y}'$ of the m points.
3. Let δ_i be the vector of squared distances from landmark i to all of the landmarks, and let $\delta_\mu = (\delta_1 + \ldots + \delta_m)/m$. Let $L^\sharp = [e_1/\sqrt{\lambda_1},\ldots,e_d/\sqrt{\lambda_d}]^T$
4. Now for a point x which is not a landmark, if δ_x is its vector of squared distances to the m landmarks, we embed x as $y = -\frac{1}{2}L^\sharp(\delta_x - \delta_\mu)$.

Kernel PCA Equivalence The matrix $\mathbf{B}$ obtained can be used as the kernel matrix $\mathbf{K}$ in Kernel PCA, as long as $\mathbf{B}$ is PSD. [MRT18]

2.5 Isomap

Definition This method simply combines what we have described above [TSL00]:

1. Build a weighted graph from the data (1.9)
2. Compute a distance matrix of shortest-path distances (1.10.1)
3. Apply classical MDS to the distance matrix to obtain the embedding (2.4.2)

Kernel PCA Equivalence The matrix $\mathbf{B}$ obtained in the use of MDS in (3) can be used as the kernel matrix $\mathbf{K}$ in Kernel PCA, as long as $\mathbf{B}$ is PSD. [MRT18]

Considerations The resulting embedding is highly sensitive to the choices made when doing step 1.

2.6 Laplacian Eigenmaps

Definition This method is a relatively simple combination of things discussed above:

1. Build an unweighted graph from the data (1.9)
2. Compute a similarity matrix $\mathbf{W}$ from the graph (1.10.2 or 1.10.3)
3. Compute the Laplacian $\mathbf{L}$ from $\mathbf{W}$.
4. Compute the bottom k eigenvectors $e_1, \ldots, e_k$ of $\mathbf{L}$ after discarding the 0-eigenvalue.
5. The new embedded data is the $n \times k$ matrix $X' = [e_1, \ldots, e_k]$

Computation Standard matrix algorithms can be used for these. $\mathbf{W}$ and $\mathbf{L}$ can be stored and worked with as sparse matrices, as the $\mathbf{W}$ created will naturally be sparse in a graph which is not fully-connected.

Kernel PCA Equivalence If we let $\mathbf{L}^\dagger$ be the pseudoinverse of $\mathbf{L}$, $\mathbf{L}^\dagger$ can be used as the kernel matrix $\mathbf{K}$ in Kernel PCA, as long as $\mathbf{L}^\dagger$ is PSD. To make Kernel PCA with this $\mathbf{K}$ equivalent to Laplacian Eigenmaps we must scale the output dimensions to have unit variance. [MRT18]

Considerations The resulting embedding is highly sensitive to the choices made when doing steps 1 and 2.

2.7 Locally Linear Embedding (LLE)

Definition The intuition here is that each point on a manifold can be represented as a linear combination of its nearest neighbors, and an embedding in lower-dimensional space should have the same linear combination for each point.

1. Compute the k nearest neighbors of each point x_i, or use an ϵ-ball.
2. Compute W_{ij} that minimizes:

$$\sum_{i=1}^{n} \left|\left| x_i - \sum_{j=1}^{n} W_{ij} x_j \right|\right|_2^2$$

   subject to $\sum_j W_{ij} = 1$, and $W_{ij} = 0$ if x_j is not a neighbor of x_i (or is not in its ϵ-ball).
3. Compute d-dimensional coordinates y that minimize:

$$\sum_{i=1}^{n} \left|\left| y_i - \sum_{j=1}^{n} W_{ij} y_j \right|\right|_2^2$$

   subject to $\sum_j Y_{ij} = 0$ and $\mathbf{Y}^T\mathbf{Y} = I$. This is called the "reconstruction error."

Computation We follow [SR00] in our discussion.

To find the weights, we can actually find the weight for each point x_i individually. First, for a point x_i, if x_i has k neighbors (or k points in its ϵ-ball) $\eta_1, \dots, \eta_k$, let's define the $k \times k$ local covariance matrix $\mathbf{C}^{(L)}$ as:

$$C^{(L)}_{lm} = \langle x_i - \eta_l, x - \eta_m \rangle$$

Some texts refer to this as the **Gram Matrix** $\mathbf{G}_i$, as it is a matrix of inner products of a set of vectors which depends on x_i.

The jth weight w_j for x_i will be:

$$w_j = \frac{\sum_m C^{(L),-1}_{jm}}{\sum_{lm} C^{(L),-1}_{lm}}$$

It is actually more efficient to solve the system of linear equations:

$$\sum_l C^{(L)}_{lm} w_m = 1$$

and rescale so that $\sum_m w_m = 1$.

Now, the $n \times n$ weight matrix $\mathbf{W}$ be such that the ith row $\mathbf{W}_{i*}$ is defined as:

$$\mathbf{W}_{i*} = \begin{bmatrix} 0 & 0 & w_1^{(x_i)} & 0 & \cdots & w_k^{(x_i)} & 0 \end{bmatrix},$$

where the nonzero entries appear at the indices of the neighbors of x_i. To find Y, we can compute:

$$M = (\mathbf{I}_n - \mathbf{W})^T (\mathbf{I}_n - \mathbf{W})$$

and take the d eigenvectors $e_1, \dots, e_d$ corresponding to the d smallest eigenvalues excluding the smallest one. Taking $\mathbf{Y} = [e_1, \dots, e_d]$, the rows of $\mathbf{Y}$ are the new data points.

Kernel PCA Equivalence If we let λ be the largest eigenvalue of $\mathbf{M}$, $\lambda \mathbf{I}_n - \mathbf{M}$ can be used as the kernel matrix $\mathbf{K}$ in Kernel PCA. [MRT18]

3 Clustering

3.1 Definition

A clustering of X made up of k clusters is a collection of sets $C = \{C_i\}_{i=1}^k$, where C forms a partition of X ($C_i \subseteq X$, $i \neq j \Rightarrow C_i \cap C_j = \emptyset$, $\bigcup_i C_i = X$).

3.2 Centroid methods

These methods seek to find a set of points $c_1, \dots, c_k$, and from these a collection of clusters $\{C_i\}_{i=1}^k$ with:

$$C_i = \{x \in X | \forall j \; d(x, c_i) \leq d(x, c_j)\},$$

such that one of these objectives is minimized:

$$\frac{1}{2} \sum_{i=1}^k \sum_{x,y \in C_k} d(x, y) \overset{d=d^2_{\mathbb{R}}}{=} \sum_{i=1}^k n_k \sum_{x \in C_i} d(x, c_i) \quad \text{[HTFF09]}$$

$$\sum_{i=1}^k \sum_{x \in C_i} d(x, c_i) \quad \text{[Dau17]}$$

The first objective given, which is the sum of all inter-point distances in the clusters, is called **within-point scatter**. Where centroid methods differ is in the distance or dissimilarity d used and in restrictions on $c_1, \ldots, c_k$. These impact what different algorithms can be used to solve the problem or approximate the solution.

3.2.1 k-means

Distance: Euclidean distance is used for $d(x, c_i)$ (1.3.5)
Centroids: No restrictions on $c_1, \ldots, c_k$

Computation An approximation algorithm for this is what is commonly known as the "k-means clustering algorithm."

1. Initialize randomly chosen cluster centers $\{c_i\}_{i=1}^k$. There are many ways to do this. One is to randomly choose points in the data.
2. Until cluster assignments don't change:
   (a) Assign each x to the cluster C_i with the nearest corresponding c_i.
   (b) Compute c_i as the mean of cluster C_i.

Considerations The k-means clustering algorithm will not necessarily find a globally optimal solution. In practice, we can run it many times and choose the resulting clustering which minimizes the objective [Dau17]. The algorithm can also run into cases where, in step 2(b), cluster C_i is empty, in which case c_i can be chosen randomly.

3.2.2 k-medians

Distance: Manhattan distance is used for $d(x, c_i)$ (1.3.4)
Centroids: No restrictions on $c_1, \ldots, c_k$.

Computation An approximation algorithm for this which is very similar to the k-means algorithm can be used:

1. Assign each x to the cluster C_i with the nearest (Manhatan distance) corresponding c_i. (can initialize with randomly chosen c_i's)
2. Compute c_i as the median of cluster C_i (in Euclidean spaces with more than one dimension, the "median" of a set of points can be computed using the median along each axis)

3.2.3 k-medoids

Distance: Any distance metric can be used for $d(x, c_i)$
Centroids: $c_1, \ldots, c_k$ must be points in the data.

Computation This can be approximated by repeatedly swapping some c_i with a point in the data and seeing if it reduces the cost defined above. Alternatively, we can use an algorithm similar to k-means: [HTFF09]

1. Initialize randomly chosen cluster assignment.
2. Until cluster assignments don't change:
   (a) Let the center c_i of cluster C_i be $\operatorname{argmin}_{x \in C_i} \sum_{y \in C_i} d(x, y)$
   (b) Assign each x to the cluster C_i with the nearest (using d) cluster center.

3.3 Agglomerative/Bottom-up Hierarchical Clustering

3.3.1 General Method

These methods are all fundamentally the same. They rely on repeatedly merging the two clusters which are "closest."

1. Start with a collection of clusters $\{C_i\}_i = \{\{x_i\}\}_i$
2. Until the collection of clusters is $\{X\}$:
   (a) Let A, B be the clusters in $\{C_i\}_i$ minimizing $d(A, B)$
   (b) Merge A and B. i.e. $\{C_i\}_i \leftarrow (\{C_i\}_i - \{A, B\}) \cup \{A \cup B\}$.

The distinction between them is how "closeness" $d(A, B)$, a.k.a. the "linkage criterion" is defined and certain computational improvements which can be made for some definitions of closeness. We follow [LLSE11] in our discussion of these.

3.3.2 Dendrograms

When agglomerative hierarchical clustering is performed, we may create a "dendrogram" by making a tree recording how the clusters joined over time, with each node being a cluster merger. The altitude of the node for A and B joining is $d(A, B)$. A clustering is considered *stable* if it covers a large range of altitudes, and *unstable* otherwise.

3.3.3 Lance and Williams method

Many of the algorithms described are instances of the general Lance and Williams (LW) method, where the distance between a cluster C and a cluster $A \cup B$ formed by merging clusters A and B is given by:

$$d(A \cup B, C) = \alpha_A d(A, C) + \alpha_B d(B, C) + \beta d(A, B) + \gamma |d(A, C) - d(B, C)|$$

When this can be used for a given d, agglomerative hierarchical clustering can be done in $O(n^2 \log(n))$ [LLSE11].

3.3.4 Minimum/Single-Linkage clustering (SLC)

$$d(A, B) = \min_{a \in A, b \in B} d(a, b)$$

The LW parameters are:

$$\alpha_A = \alpha_B = 1/2,\ \beta = 0,\ \gamma = -1/2,$$

giving the recurrence relation:

$$d(A \cup B, C) = \min(d(A, C), d(B, C)).$$

We can also think about the clustering at the point where all x and y with $d(x, y) \leq \epsilon$ are in the same cluster (which is achieved when we've only recursively merged up to the point where $d(A, B)$ is still $< \epsilon$). This clustering is the same as the connected components of the ϵ-ball graph (1.9.2).

Considerations A thin chain of points between two real clusters will make them one cluster in Single-Linkage clustering, making SLC not very robust to noise.

3.3.5 Maximum/Complete-Linkage clustering

$$d(A, B) = \max_{a \in A, b \in B} d(a, b)$$

The LW parameters are:

$$\alpha_A = \alpha_B = 1/2,\ \beta = 0,\ \gamma = 1/2,$$

giving the recurrence relation:

$$d(A \cup B, C) = \max(d(A, C), d(B, C)).$$

3.3.6 Unweighted average linkage clustering (UPGMA)

"Unweighted Pair Group Method with Arithmetic mean."

$$d(A, B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a, b)$$

The LW parameters are:

$$\alpha_A = \frac{|A|}{|A| + |B|},\ \alpha_B = \frac{|B|}{|A| + |B|},\ \beta = 0,\ \gamma = 0,$$

giving the recurrence relation:

$$d(A \cup B, C) = \frac{|A|}{|A| + |B|} d(A, C) + \frac{|B|}{|A| + |B|} d(B, C).$$

This is called "unweighted" because each point ends up getting the same weight when the distances are averaged.

3.3.7 Weighted average linkage clustering (WPGMA)

"Weighted Pair Group Method with Arithmetic mean." There is actually no way to write $d(A, B)$, as it depends on the sequence of merges which constructed A and B. WPGMA can, however, be expressed with the LW parameters:

$$\alpha_A = \alpha_B = \frac{1}{2},\ \beta = 0,\ \gamma = 0,$$

giving the recurrence relation:

$$d(A \cup B, C) = \frac{1}{2} d(A, C) + \frac{1}{2} d(B, C).$$

This is called "weighted" because points in smaller clusters end up getting a higher weight when the distances are averaged.

3.3.8 Unweighted centroid linkage (UPGMC)

If we call the centroid of cluster A c_A, and that of cluster B c_B, then:

$$d(A, B) = d(c_A, c_B)$$

This can't be expressed in LW parameters, but with some reasoning we can get the recurrence relation that:

$$c_{A \cup B} = \frac{|A|}{|A| + |B|} c_A + \frac{|B|}{|A| + |B|}$$

And then:

$$d(A \cup B, C) = d(c_{A \cup B}, c_C)$$

3.3.9 Weighted centroid linkage (WPGMC)

If we call the centroid of cluster A c_A, and that of cluster B c_B, then:

$$d(A, B) = d(c_A, c_B)$$

Now, when we join clusters, rather than taking the true centroid we take the simple mean of the centroids of the two clusters (this is similar to the distinction between UPGMA and WPGMA):

$$c_{A\cup B} = 0.5c_A + 0.5c_B$$

And then:

$$d(A \cup B, C) = d(c_{A\cup B}, c_C)$$

Like UPGMC this can't be expressed with LW parameters.

3.4 Divisive/Top-down Hierarchical Clustering

Top-down approaches to hierarchical clustering start with one cluster containing all of the points, then progressively break it up.

3.4.1 diana

The name of this method is short for DIvisive ANAlysis clustering. Here is its algorithm:

1. Start with all points in one cluster, i.e. with the clustering $C = \{X\}$.
2. Until we have reachced the desired number of clusters (or each point is in its own cluster):
   (a) Pick a cluster $C_i \in C$ (say, the largest one)
   (b) Find the point x whose average distance to all points in C_i is maximized.
   (c) Create a cluster $A = \{x\}$ and $B = C_i - \{x\}$.
   (d) Until there are no such points, add points to A which have the highest positive difference:
   $$[\text{Average distance to } B] - [\text{Average distance to } A]$$
   (e) Let $C = C \cup \{A\} \cup \{B\} - \cup\{C_i\}$

3.5 Generative/Distribution-based Clustering

In generative clustering methods, we define some mixture model M (which is a mixture over k different submodels $M_1, \ldots, M_k$) of how our data are generated:

$$x_1, \ldots, x_n \overset{iid}{\sim} M = \begin{cases} M_1 & \text{w.p. } \pi_1 \\ \vdots & \\ M_k & \text{w.p. } \pi_k \end{cases}.$$

After fitting the model to the data (i.e. choosing parameters θ_i for each M_i and priors π_i) to maximize likelihood:

$$\boldsymbol{\theta}, \boldsymbol{\pi} = \arg\max_{\boldsymbol{\theta},\boldsymbol{\pi}} P(x_1, \ldots, x_n | M, \boldsymbol{\theta}, \boldsymbol{\pi}),$$

we assign each point to the part of the mixture model which assigns it the highest probability, hence getting that the ith cluster C_i is:

$$C_i = \{x \in X | \forall j \ \pi_i P(x|M_i, \boldsymbol{\theta}) \geq \pi_j P(x|M_j, \boldsymbol{\theta})\}$$

3.5.1 Gaussian mixture

Definition Here M_i is $N(\mu_i, \Sigma_i)$ and $\boldsymbol{\theta}$ is $\{\mu_i, \Sigma_i\}_i$.

Computation It is hard to maximize the likelihood here exactly, but we can use an Expectation-Maximization algorithm to find a local maximum:

1. Initialize $\boldsymbol{\theta}$ randomly, $\boldsymbol{\pi}$ uniform. Initialize the assignment weight of x_i to cluster j $w_j^{(i)}$ as $1/k$.
2. Until $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$ converge:
   (a) (Expectation): For all $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$, let:

$$w_j^{(i)} = \pi_j \sqrt{\det(\Sigma_j^{-1})} \exp\left(-0.5(x_i - \mu_j)^T \Sigma_j^{-1}(x_i - \mu_j)\right)$$

   And normalize so that $\sum_j w_j^{(i)} = 1$ for all i.
   (b) (Maximization): For all j, set:

$$n_j = \sum_{i=1}^{n} w_j^{(i)}; \quad \pi_j = \frac{n_j}{n}; \quad \mu_j = \frac{1}{n_j} \sum_{i=1}^{n} w_j^{(i)} x_i; \quad \Sigma_j = \frac{1}{n_j} \sum_{i=1}^{n} w_j^{(i)} (x_i - \mu_j)(x_i - \mu_j)^T$$

Considerations The EM algorithm will attain a local maximum, and not necessarily a global one. The EM algorithm can also be used to obtain soft cluster assignments, in the form of the normalized weights $w_j^{(i)}$.

3.6 Joint Methods

There are some clustering methods which, technically, just apply a dimension-reduction method followed by a clustering method. In principle, arbitrary combinations can be used, and there's nothing to stop someone from doing e.g. Isomap followed by Gaussian mixture. Though difficulties do arise when combining very hyperparameter-sensitive dimension-reduction methods with very hyperparameter-sensitive clustering methods.

3.6.1 Spectral Clustering

Spectral clustering, as typically used, is the following:

1. Perform a Laplacian Eigenmap embedding of the data (2.6)
2. Apply k-means to the embedded data (3.2.1)

Considerations As with the Laplacian Eigenmap embedding, there are many choices which will affect the output. Here our results will also be affected by what dimension d we decide to embed our data in, and what k we choose for the number of clusters.

References

[BG05] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[CLRS22] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[Dau17] Hal Daumé. *A course in machine learning*. 2017.

[DG03] Sanjoy Dasgupta and Anupam Gupta. "An elementary proof of a theorem of Johnson and Lindenstrauss". In: *Random Structures & Algorithms* 22.1 (2003), pp. 60–65.

[DT04] Vin De Silva and Joshua B Tenenbaum. *Sparse multidimensional scaling using landmark points*. Tech. rep. technical report, Stanford University, 2004.

[EMTB00] F Esposito, D Malerba, V Tamma, and HH Bock. "Similarity and Dissimilarity". In: *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data*. Springer. 2000, pp. 139–197.

[Gad10] Kenneth Gade. "A Non-singular Horizontal Position Representation". In: *Journal of Navigation* 63.3 (May 2010), pp. 395–417. DOI: 10.1017/s0373463309990415.

[HTFF09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[IM07] Jordi Inglada and Grgoire Mercier. "A new statistical similarity measure for change detection in multitemporal SAR images and its extension to multiscale change analysis". In: *IEEE transactions on geoscience and remote sensing* 45.5 (2007), pp. 1432–1445.

[LHC06] Ping Li, Trevor J Hastie, and Kenneth W Church. "Very sparse random projections". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 287–296.

[LLSE11] Sabine Landau, Morven Leese, Daniel Stahl, and Brian S Everitt. *Cluster analysis*. John Wiley & Sons, 2011.

[MGL22] Patrick Mair, Patrick JF Groenen, and Jan de Leeuw. "More on multidimensional scaling and unfolding in R: smacof version 2". In: *Journal of Statistical Software* 102 (2022), pp. 1–47.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[Nav01] Gonzalo Navarro. "A guided tour to approximate string matching". In: *ACM computing surveys (CSUR)* 33.1 (2001), pp. 31–88.

[RB19] Raúl Rabadán and Andrew J Blumberg. *Topological data analysis for genomics and evolution: topology in biology*. Cambridge University Press, 2019.

[SR00] Lawrence K Saul and Sam T Roweis. "An introduction to locally linear embedding". In: *unpublished. Available at: http://www.cs.toronto.edu/~roweis/lle/publications.html* (2000).

[TSL00] Joshua B Tenenbaum, Vin de Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500 (2000), pp. 2319–2323.

[Tsy08] A. Tsybakov. *Introduction to Nonparametric Estimation*. 2008.

[Was17] L Wasserman. *Optimal transport and wasserstein distance*. 2017.

[Wic03] Florian Wickelmaier. "An introduction to MDS". In: *Sound Quality Research Unit, Aalborg University, Denmark* 46.5 (2003), pp. 1–26.