29

UNSUPERVISED STATISTICAL LEARNING

29.1 Singular value decomposition (SVD) and matrix factorization

29.1.1 SVD theory

29.1.1.1 *SVD fundamentals*

Unsupervised learning aims to discover the structures of input data. Major areas of unsupervised learning includes dimensional reduction and clustering. We start with singular value decomposition, which directly describes the decomposition of data matrix that reveals latent structures. The SVD theory covered in the following are largely adapted from section 4.9.

Theorem 29.1.1 (complete form SVD). *Any matrix $A \in \mathbb{R}^{m\times n}$ has a factorization given by [Figure 29.1.1]*

$$A = U\Sigma V^T$$

*where $U \in \mathbb{R}^{m\times m}, V \in \mathbb{R}^{n\times n}, \Sigma \in \mathbb{R}^{m\times n}$ and Σ is rectangle diagonal matrix. The diagonal entries in Σ, known as **singular values**, consist of first $r = rank(A)$ **non-zero, positive, decreasing entries** $(\sigma_1, ..., \sigma_r)$, and other zeros.*
Moreover, σ_i^2 is a eigenvalue of matrix AA^T and A^TA; u_i and v_i (columns in U and V) are eigenvectors of AA^T and A^TA, respectively.

Proof. See Theorem 4.9.1. □

Note 29.1.1 (interpretation of blocks).

- The first r columns of U span the range space of A, i.e., $\mathcal{R}(A)$, the last $n-r$ columns span $\mathcal{R}(A)^\perp$, and by fundamental theorem of linear algebra, $\mathcal{R}(A)^\perp = \mathcal{N}(A^T)$.
- The first r columns of V span a subspace that will contribute to the final result (we denote it as $\mathcal{N}(A)^\perp$, and by fundamental theorem of linear algebra $\mathcal{N}(A)^\perp = \mathcal{R}(A^T)$), while the last $n-r$ columns span the null space $\mathcal{N}(A)$, which will not contribute to the final result.
- The transformation $y = Ax = U\Sigma V^T x$ can be interpreted in the SVD framework: first map/decompose the vector x into components lying in two subspaces (one space will contribute, and one null space that will not contribute), then only scale the components in the contributing space; finally the scaled components are recovered in the range space of A spanned by the first r columns in U.

Remark 29.1.1 (redundant information in full form SVD).

- If we change the entries in the last $n-r$ columns of V, the resulting matrix from product $U\Sigma V^T$ will not change.
- If we change the entries in the last $m-r$ columns of U, the resulting matrix from product $U\Sigma V^T$ will not change.

Remark 29.1.2 (relationship between U and V). It is a common mistake to think that U and V are orthogonal to each other, i.e. $U^TV = I$. Actually, U and V are orthogonal to each other when A is symmetric. Particularly, we have:

- U consists of the eigenvectors of AA^T, and V consists of the eigenvectors of A^TA.
- If A is not square, U and V cannot even multiply together (incompatible sizes).
- If A is symmetric, columns in U and V are eigenvectors of A^2 and A. Therefore, U and V are orthogonal to each other.

Corollary 29.1.1.1 (compact form SVD). *Any matrix $A \in \mathbb{R}^{m\times n}$ has a factorization given by [Figure 29.1.1]*

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T = U_r \Sigma_r V_r^T$$

where $U_r \in \mathbb{R}^{m\times r}, V_r \in \mathbb{R}^{r\times n}, \Sigma_r \in \mathbb{R}^{r\times r}$ and $\Sigma = diag(\sigma_1, ..., \sigma_r)$ is diagonal matrix, and the diagonal entries being non-zero/positive decreasing entries.Moreover, $\sigma_i^2 = \lambda_i(AA^T) = \lambda_i(A^TA)$ and u_i and v_i are eigenvectors of A^TA and AA^T.

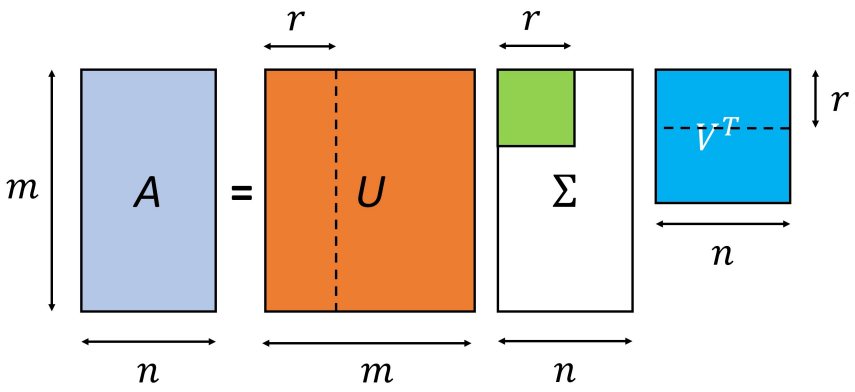
Lemma 29.1.1 (SVD of inverse). *Let A be a invertible matrix with SVD as*

$$A = U\Sigma V^T$$

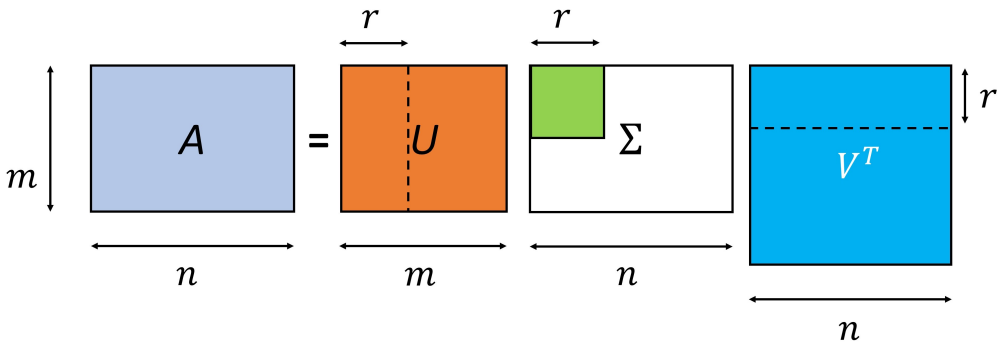
then

$$A^{-1} = V\Sigma^{-1}U^T$$

Proof. $A^{-1} = (U\Sigma V^T)^{-1} = V^{-T}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^T$. □



(a) Demonstration of SVD for a tall and skinny matrix.



(b) Demonstration of SVD for a short and fat matrix.

Figure 29.1.1: Demonstration of SVD for matrices of two different shapes. The dashed lines highlight the compact form SVD.

29.1.1.2 *SVD and matrix norm*

The singular values from SVD are closely related to Frobenius norm and 2-norm.

Theorem 29.1.2 (Frobnius norm). *For any matrix $A \in \mathbb{R}^{m \times n}$, then*

$$\|A\|_F^2 = \sum_{i=1}^{r} \sigma_i^2$$

where σ_i are singular values of A.

Proof. See Theorem 4.9.2 □

Theorem 29.1.3 (matrix 2-norm). *For any matrix $A \in \mathbb{R}^{m \times n}$, then*

$$\|A\|_2^2 = \sigma_1^2$$

or

$$\|A\|_2 = \sigma_1$$

where σ_1 is the largest singular value of A.

Particularly, if A is symmetric, then

$$\|A\|_2 = \max_i |\lambda_i| .$$

Proof. See Theorem 4.9.3. □

29.1.1.3 *SVD low rank approximation*

This section covers the SVD approach to matrix low rank approximation in terms of Frobenius norm and 2-norm.

Theorem 29.1.4 (Frobenius norm low rank approximation). *Let $A \in \mathbb{R}^{m \times n}$, with $rank(A) = r$, then minimization problem*

$$\min_{A_k \in \mathbb{R}^{m \times n}, rank(A_k) = k} \|A - A_k\|_F^2$$

with $1 \leq k \leq r$ has the solution

$$A_k^* = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

with the optimal value of $\sum_{i=k+1}^{r} \sigma_i^2$

Proof. See Theorem 4.9.4. □

Corollary 29.1.4.1 (rank approximation alternative formulation). *Let S be a matrix of size $m \times n$. Let S have SVD given by*

$$S = U \Sigma V^T.$$

It follows that

- *the value of*
$$\|S - P\|_F^2 = Tr((S-P)(S-P)^T)$$
is minimum among matrices P of the same size but of rank $r \leq rank(S)$, when $P = U_r U_r^T S$, where U_r is $m \times r$ and the columns of U_r are the r normalized eigenvectors of SS^T with the r largest eigenvalues (or the first r columns of U).
- *Alternatively, $P = SV_r V_r^T$, where V_r is $r \times n$ and the columns of V_r are the r normalized eigenvectors of $S^T S$ with the r largest eigenvalues (or the first r columns of V).*

Proof. Note that
$$P = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$
□

Lemma 29.1.2. *For any matrix $A \in \mathbb{R}^{n \times d}$, let $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T, k \leq rank(A) = r$, then*
$$\|A - A_k\|_2 = \sigma_{k+1}$$

Proof. Let $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ be the SVD of A, then we have
$$A - A_k = \sum_{i=1+k}^{r} \sigma_i u_i v_i^T$$
Based on the definition of 2-norm, we have
$$\|A - A_k\|_2^2 = \max_{\|x\|=1} \|(A - A_k)x\|_2^2$$
In order to maximize the above, x should lie in the subspace spanned by $v_{k+1}, v_{k+2}, ..., v_r$, and we write $x = \sum_{i=k+1}^{r} a_i v_i$ then we have
$$\|(A - A_k)x\|_2^2 = \sum_{i=1+k}^{r} a_i^2 \sigma_i^2 \leq \sigma_{k+1}^2 \sum_{i=k+1}^{r} a_i^2 = \sigma_{k+1}^2$$
and the maximum is attained at $x = v_{k+1}$ □

Theorem 29.1.5 (matrix 2-norm low rank approximation). *Let $A \in \mathbb{R}^{m \times n}$, with $rank(A) = r$, then minimization problem*
$$\min_{A_k \in \mathbb{R}^{m \times n}, rank(A_k) \leq k} \|A - A_k\|_2^2$$

with $1 \leq k \leq r$ has the solution

$$A_k^* = \sum_{i=1}^{k} \sigma_k u_i v_i^T$$

and

$$\left\| A - A_k^* \right\|_2 = \sigma_{k+1}^2$$

Proof. See Theorem 4.9.5. □

29.1.2 Principal component analysis (PCA)

29.1.2.1 *Statistical perspective of PCA*

Suppose we are given a set of sample points $x_1, ..., x_n, x_i \in \mathbb{R}^D$, and our goal is to find low-dimensional projected sample points $y_1, ..., y_n, y_i \in \mathbb{R}^d, , d \ll D$ via $y_i = U^T x_i, U \in \mathbb{R}^{D \times d}$ such that the variations of in $\{x_1, ..., x_n\}$ are maximally preserved [Figure 29.1.2]. The column vectors $u_1, ..., u_d \in \mathbb{R}^D$ are known as **principal component directions or principal components**, and the transformed sample point (a vector) y_i is known as **principal component scores**, with k component given by $u_k^T x_i$.

The goal of preserving sample variation can be formulated as the following optimization problem. Given a set of multi-dimensional sample point $x_1, ..., x_N \in \mathbb{R}^D$ with sample covariance matrix S defined by

$$S = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \overline{x})(x_i - \overline{x})^T.$$

The d sample principal components are d unit vectors $u_i, u_i \in \mathbb{R}^D, i = 1, 2, ..., D, U = [u_1, ..., u_D]$ satisfying

$$\begin{aligned}
u_1 &= \arg\max_u u^T S u, \ s.t. \ u^T u = 1 \\
u_2 &= \arg\max_u u^T S u, \ s.t. \ u^T u = 1, u^T u_1 = 0 \\
&\dots \\
u_d &= \arg\max_u u^T S u, \ s.t. \ u^T u = 1, u^T u_2 = 0, ..., u^T u_{d-1} = 0
\end{aligned}$$

Figure 29.1.2: Principal components for 2D samples.

It turns out that the optimization problems are indeed the top d eigenvectors of Σ_X, as we show in the following theorem.

Theorem 29.1.6 (principal components are top eigenvectors). *The principal components vectors $u_1, ..., u_d$ are given by the top d eigenvectors of Σ_X.*

Proof. (1) Use Reyleigh quotient theorem [Theorem 4.8.4] the top eigenvector of Σ_X maximize $u^T \Sigma_X u$ under the constraint $u^T u = 1$. (2) Use quadratic form maximization theorem [Theorem 4.12.4], we know that $u_1, ..., u_d$ are indeed the top eigenvectors. □

In addition, there are a number of critical properties regarding principal components and principal component scores.

Theorem 29.1.7. *Let $x_1, x_2, ..., x_N \in \mathbb{R}^D$ be a set of random samples. Let U of the matrix whose columns are the top d principal components of sample covariance matrix S. Let $\lambda_1, ..., \lambda_d$ be the top d eigenvalues. The principal component scores are given by $y_i = U^T x_i$. It follows that*

- *The sample covariance matrix Σ_y of $y_1, ..., y_N, y_i \in \mathbb{R}^d$ are diagonal. The diagonal terms are given by λ_i.*

- *The total variance of principal component scores is*

$$\sum_{i=1}^{N}(y_i - \overline{y})^T(y_i - \overline{y}) = (N-1)(\lambda_1 + \lambda_2 + \cdots \lambda_d),$$

where total variance of the original sample is

$$\Delta^2 = \sum_{i=1}^{N}(x_i - \overline{x})^T(x_i - \overline{x}) = (N-1)(\lambda_1 + \lambda_2 + \cdots \lambda_D).$$

Proof. (1) First note that

$$\begin{aligned}\frac{1}{N-1}\sum_{i=1}^{N}(y_i - \overline{y})(y_i - \overline{y})^T &= \sum_{i=1}^{N}(U^T x_i - U^T\overline{x})(U^T x_i - U^T\overline{x})^T \\ &= \frac{1}{N-1}\sum_{i=1}^{N} U(x_i - \overline{x})^T(x_i - \overline{x})^T U^T \\ &= USU^T\end{aligned}$$

To see the off diagonal terms are zero, we have $e_i^T USU^T e_j = u_i^T S u_j = \lambda_j u_i^T u_j = 0, i \neq j$. To see the diagonal terms, we have To see the off diagonal terms are zero, we have $e_i^T USU^T e_i = u_i^T S u_i = \lambda_i u_i^T u_i = \lambda_i$. To see the diagonal terms, we have

(2)

$$
\begin{aligned}
&\sum_{i=1}^{N}(y_i-\overline{y})^T(y_i-\overline{y}) \\
&=\sum_{i=1}^{N}(U^Tx_i-V^T\overline{x})^T(U^Tx_i-U^T\overline{x}) \\
&=\sum_{i=1}^{N}(x_i-\overline{x})^TUU^T(x_i-\overline{x}) \\
&=\sum_{i=1}^{N}Tr((x_i-\overline{x})^TUU^T(x_i-\overline{x})) \\
&=\sum_{i=1}^{N}Tr(UU^T(x_i-\overline{x})(x_i-\overline{x})^T) \\
&=Tr(UU^T\sum_{i=1}^{N}(x_i-\overline{x})(x_i-\overline{x})^T) \\
&=(N-1)Tr(UU^TS) \\
&=Tr(U^TU\Lambda U^TU) \\
&=Tr(\Lambda) \\
&=\lambda_1+\lambda_2+\cdots\lambda_j
\end{aligned}
$$

where we use matrix trace cyclic property [Lemma A.8.8). □

Remark 29.1.3 (rank deficiency for high dimensional input data). When the input data $x_i \in \mathbb{R}^D$ is high dimensional such that $D \gg N$, the sample covariance matrix

$$S=\frac{1}{N-1}\sum_{i=1}^{N}(x_i-\overline{x})(x_i-\overline{x})^T,$$

will have rank at most $N-1$ (when columns are linearly independent) or smaller (when there are linearly dependent columns).

29.1.2.2 *Geometric fundamentals of PCA*

Consider a set of points $X=\{x_1,x_2,...,x_N\}, x_i \in \mathbb{R}^D$ and assume they are given by

$$x_i=\mu+U_dy_i+\epsilon_i,$$

where $\mu \in \mathbb{R}^D, U_d \in \mathbb{R}^{D\times d}$ is a matrix with independent columns, $y_i \in \mathbb{R}^d$ is the linear combination coefficients, and $\epsilon_i \in \mathbb{R}^D$ is the additional noise.

The geometric picture of such representation is that data points are approximately lying on a low-dimensional affine space, characterized by shift μ and subspace basis U_d. The goal principal component analysis is to find $\mu, U_d, \{y_i\}$, when d is given, such that the sum of squared errors is minimized.

Remark 29.1.4 (redundancy in the representation). [1, p. 19]

- They are redundancy in the above representation because of the arbitrariness in the choice of μ and U. For example, $x_i = \mu + U_d y_i = (\mu + U_d y_0) + U_d(y_i - y_0)$. We can remove this translational ambiguity by requiring $\mu = \frac{1}{N}\sum_{i=1}^N x_i$; therefore we are effectively dealing with demeaned data.
- Another ambiguity is due to the arbitrariness in the choice of basis spanning the subspace. We can remove this ambiguity by enforcing orthonormality in the columns of U_d.

The principal component problem can be solved by the following optimization framework.

Lemma 29.1.3 (geometric PCA in the optimization framework). *Consider a set of points $X = \{x_1, x_2, ..., x_N\}, x_i \in \mathbb{R}^D$ and further assume they are demeaned such that $\frac{1}{N}\sum_{i=1}^N x_i = 0$.*

- *Then finding d orthonormal basis (i.e. U_d) and $y_i, i = 1, ..., N$ that minimizes the sum of squared errors is given by*

$$\min_{U_d, \{y_i\}} \|X - U_d Y\|_F^2 = \sum_{i=1}^N \|x_i - U_d y_i\|^2, s.t., U_d^T U_d = I_d.$$

- *The optimization problem can be alternatively formulated as*

$$\min_{U_d} \left\|X - U_d U_d^T X\right\|_F^2, s.t., U_d^T U_d = I_d.$$

- *The necessary condition for $y_i, i = 1, ..., N$ to achieve optimality is*

$$\hat{y}_i = U_d^T x_i.$$

Proof. The Lagrangian function is given by

$$L = \sum_{i=1}^N \|x_i - U_d y_i\|^2 + Tr((I_d - U^T U)\Lambda),$$

where $\Lambda \in \mathbb{R}^{d \times d}$ is the matrix of Lagrange multipliers. Then we have

$$\frac{\partial L}{\partial y_i} = 0 \implies -2U_d^T(x_i - U_d y_i) = 0 \implies y_i = U_d^T x_i,$$

where we use the fact that $U_d^T U_d = I_d$. □

Theorem 29.1.8 (PCA via SVD). *[1, p. 21]Let $X = [x_1, x_2, ..., x_N] \in \mathbb{R}^{D \times N}$ be the data matrix. Let $X = U\Sigma V^T$ be the singular value decomposition (SVD) of X. Then for any given $d < D$, a solution to PCA is the first d columns of$U$, given as $U_d = [u_1, u_2, ...u_d]$ and $\{y_i\}$ is the top $d \times N$ sub matrix $\Sigma_d V_d^T$ of the matrix ΣV^T (each column of length d is one y_i and in y_i each row is scaled in $\sqrt{\lambda_i}$).*

Proof.

$$\begin{aligned}\left\|X - UU^T X\right\|_F^2 &= Tr((X - UU^T X)^T (X - UU^T X)) \\ &= Tr(X^T X) - Tr(XUU^T X) - Tr(UU^T X^T X) + Tr(X^T UU^T UU^T X) \\ &= Tr(X^T X) - 2Tr(XUU^T X) + Tr(X^T UU^T X) \\ &= Tr(X^T X) - Tr(X^T UU^T X)\end{aligned}$$

Note that $Tr(XUU^T X) = Tr(U^T XX^T U) = \sum_{i=1}^d u_i^T XX^T u_i$. From Rayleigh quotient theorem [Theorem 4.8.4], we know that maximum of $Tr(U^T XX^T U$ is attained when u_i are the top d eigenvectors. □

Remark 29.1.5 (pitfalls for statistical approach and geometrical approach). Let X be the data matrix $X \in \mathbb{R}^{p \times N}$. In statistical approach, we calculate principal components by eigen-decomposition or SVD from sample covariance matrix of X, in which X **is demeaned**.

In the geometrical approach, if we directly perform SVD on X without demeaning X, we will get different results.

29.1.2.3 *Robust PCA with outliers*

PCA are results based on least square optimization, which is known to be not robust against outlier. There are some simple ways to make PCA robust, including removing outliers in the data preprocessing step and using robust objective function in the optimization step.

In this section, we look at some simple algorithms that are designed to handle outliers. For state of the art algorithms, reader can refer to [2]. The first algorithm we look at is **iteratively reweighted least square method** [3, p. 105]. The weights of outliers,

as automatically determined by the algorithm, are adjusted to smaller values relative to weights of inliers.

Algorithm 48: Iterative reweighted least square PCA with outliers algorithm

Input: Data matrix $X \in \mathbb{R}^{D \times N}$, dimension d, and parameter $\epsilon_0 > 0$

Initialize low dimensional subspace U, mean vector μ, and projection Y from PCA.

repeat

  Compute $\epsilon_j = \left\| x_j - \mu - U y_j \right\|_2, w_j = \frac{\epsilon_0^2}{\epsilon_0^2 + \epsilon_j^2}$ for data point x_j.

  Construct weighted mean and scatter matrix

$$\mu = \frac{\sum_{j=1}^N w_j (x_j - U y_j)}{\sum_{j=1}^N w_j}, \Sigma = \frac{\sum_{j=1}^N w_j (x_j - \mu)(x_j - \mu)^T}{\sum_{j=1}^N w_j}$$

  Calculate the top eigenvectors U of Σ.

$$Y = U^T (X - \mu 1^T)$$

until *convergence of* $\mu 1^T + UY$;

Output: The low dimensional subspace U, Y, μ

The second method we look at is a random sample consensus (RANSAC) method[3, p. 107]. The core idea is to randomly pick a set of data points and perform regular PCA on the selected data. Then we check if sufficient number of data points are lying on the subspace; if not, the algorithm continues by performing PCA on a new subset of data. RANSAC usually can tolerate up to 50 percent of outliers.

Algorithm 49: Random sample consensus PCA with outliers algorithm

Input: Data set $\mathcal{X}$ consists of $x_1, x_2, ..., x_N, x_i \in \mathbb{R}^D$, maximum number of iterations k, threshold on fitting error τ, threshold on minimum number of inliers N_{min}

Initialize $i = 0$

repeat

  Get subsample cX_i from $\mathcal{X}$.

  Construct the subspace U_i by using PCA on $\mathcal{X}_i$.

  Find out $\mathcal{X}_{inlier} = \{x \in \ : dist(x, U_i) \leq \tau\}$.

  If $|\mathcal{X}_{inlier}| \geq N_{min}$, break.

  Set $i = i + 1$.

until $i = k$;

Output: The estimated subspace U_i

Now we establish some theoretical understanding on how many iteration we need to identify the subspace[3, p. 508].

Assume the given N data points contain p portion inliers and $1-p$ portion of outliers. We need to sample k points with replacement to estimate the model.

- In one sample of size k, the probability that all k samples are inlier is p^k.
- In m trials, the probability that there is at least one sample are all inliers is $1-(1-p^k)^m$.
- The number of trials needed to guarantee the probability of at least one trial samples are all inliers is at least q is

$$m \geq \frac{\log(1-q)}{\log(1-p^k)}$$

29.1.3 Sparse coding and dictionary learning

29.1.3.1 *Sparse coding*

The sparse coding problem is to find a sparse representation of the data as a linear combination of 'atoms' from a fixed, pre-computed **dictionary**. The dictionary can either from mathematical constructs such as a discrete wavelet basis or components adaptively learned from data. Usually, we aim to find the fewest atoms to represent the data as accurate as possible. Sparse codes not only have important application in data compression, but also offers tools to examine structure underlying the data. The process of learning a dictionary from data will be addressed in the next section.

The sparse coder problem is equivalent to the following optimization

$$\arg\min_y \|x - Uy\|_2^2, \quad s.t. \|y\|_0 \leq K$$

where $x \in \mathbb{R}^D$ is the original data, $U \in \mathbb{R}^{D \times F}$ is the dictionary matrix, y is the code that we are optimizing for, and K is a positive integer constraining the non-zero element count in y.

Alternatively, sparse coder problem can be viewed as seeking the sparsest code within specified tolerance Tol, i.e.,

$$\arg\min_y \|y\|_0, \quad s.t. \|x - Uy\|_2^2 \leq Tol$$

This L_0 optimization problem is known to be NP hard, but can be approximately solved via orthogonal matching pursuit algorithm (OMP) [4], Least-angle regression, Lasso, etc. OMP is an iterative greedy algorithm [algorithm 50] that from the dictionary selects

at each step one column which is most correlated with the current residuals. The algorithm then updates the residuals by projecting the observation onto the null space of the linear subspace spanned by the selected columns and the algorithm continues iteration.

Algorithm 50: Orthogonal Matching Pursuit

Input: Original data x, dictionary U with normalized columns u_i, number of nonzero coefficients K.

Initialize $k = 1$ and residual $R_k = X$.
Initialize Selected columns $U_0 = \emptyset$.
repeat
  Find u_i in columns of U with maximum inner product of $\left|\langle r_k, u_i \rangle\right|$.
  $y_k = \langle r_k, u_i \rangle$
  $U_k = U_{k-1} \cup u_k$.
  Construct orthogonal projection $P_k = U_k(U_k^T U_k)^{-1} U_k^T$.
  Update residual $r_{k+1} = (I - P_k)X$
  $k = k + 1$.
until $k > K$;

Output: coefficients $y_1, ..., y_K$

Remark 29.1.6 (using L_1 penalty to enforce sparsity)**.** Sparsity in the code can also be achieved by adding L_1 penalty. The algorithm is then given by LASSO [algorithm 32].

29.1.3.2 *Dictionary learning*

A complete dictionary learning process includes two parts: the sparse code and dictionary itself. The objective function for dictionary learning is

$$\begin{aligned} \min_{U,Y} &\|X - UY\|_F^2, \\ s.t., \forall i, &\left\|y_i\right\|_0 \leq T_0, \\ \forall j, &\left\|U_j\right\|_2^2 \leq 1, \end{aligned}$$

where $X \in \mathbb{R}^{D \times N}$ is the original data set of N samples, $U \in \mathbb{R}^{D \times F}$ is the **dictionary matrix**, $Y \in \mathbb{R}^{F \times N}$ is the **sparse code** that we are optimizing for. Note that we have the norm constraint on the component in the dictionary, which prevents learning large-valued dictionary and small-valued coding.

The optimization is non-convex. However, if we fix either U or Y and then optimize the other, it is a convex optimization problem. Therefore a common strategy to this optimization is alternating the optimization of U and Y. When we fix U and optimize Y,

we call it sparse coding stage; when we fix Y and optimize U, we call it dictionary update stage.

The sparse coding stage can be solved using methods in subsubsection 29.1.3.1. The K-SVD approach [[5], algorithm 51] to the second stage is to update each column in U one by one via SVD. Specifically, we can write

$$\begin{aligned}\|X - UY\|_F^2 &= \left\| Y - \sum_{j=1}^{K} U_j y_T^j \right\|_F^2 \\ &= \left\| \left(Y - \sum_{j \neq k} U_j y_T^j \right) - U_k x_T^k \right\|_F^2 \\ &= \left\| E_k - U_k y_T^k \right\|_F^2 \quad note\ (E_k = \left(Y - \sum_{j \neq k} U_j y_T^j \right))\end{aligned}$$

Clearly, the low-rank approximation capability of SVD allows us to use SVD to find better one rank matrix U_k, y_T to reduce the error when we fix other columns. Directly apply SVD to E_k can introduce non-zero coefficients to y_T and violate the sparse coding constraint.

Alternative, we construct the set w_k that contains the indices of examples whose k code is nonzero, i.e., $w_k = \{i | 1 \leq i \leq N, y_T^k(i) \neq 0\}$ Constructed the restricted residual by retaining only the columns whose indices are in w_k Apply SVD to E_k^R and use SVD result to update U_k and y_k (by adding zeros back).

The final algorithm is given by:

Algorithm 51: K-SVD for dictionary learning.

Input: K

Initialize dictionary matrix $U^{(0)}$ with normalized columns.

repeat

  Sparse encoding stage.

  for $n = 1,...,N$ **do**

    Draw x_n from the training data set.

    Solve sparse encoding problem via

$$\min_{y_n \in \mathbb{R}^m} \frac{1}{2} \left\| x_n - U^{(k)} y_n \right\|_2^2, s.t., \quad \left\| y_n \right\|_n \leq T_0$$

  end

  Dictionary update stage:

  for $k = 1, ..., K$ **do**

    Compute residual

$$E_k = X - \sum_{j \neq k} U_j y_{T,j}.$$

    Construct the set w_k that contains the indices of examples whose k code is nonzero, i.e., $w_k = \{i | 1 \leq i \leq N, y_T^k(i) \neq 0\}$.

    Constructed the restricted residual E_k^R by retaining only the columns whose indices are in w_k.

    Apply SVD to $E_k^R = P \Sigma Q^T$. Let U_k be the first column in P. Update $y_{T,j}$ to be the first column of Q (remember to add zeros back)multiplied by the first singular value.

  end

until *stopping criterion is met*;

Output: dictionary U and code Y.

29.1.3.3 *Online dictionary learning*

The K-SVD algorithm [algorithm 50] is a batch algorithm that usually cannot scale to very large data set. An alternative approach is online dictionary learning, where we update dictionary using data samples one at time. Given N samples, the loss function associated with the dictionary matrix while fixing sparse coding is given by[5]

$$J(U) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left\|x_i - Uy_i\right\|_2^2 + \lambda\left\|y_i\right\|_1$$
$$= \frac{1}{N}\left(\frac{1}{2}Tr\left(D^T D A_t\right) - Tr\left(D^T B_t\right)\right) + \lambda\sum_{i=1}^{t}\left\|\alpha_i\right\|_1$$

where

$$A = \sum_{i=1}^{N} y_i y_i^T, \quad B = \sum_{i=1}^{N} x_i y_i^T.$$

Using matrix trace derivative properties [Lemma A.8.9], the gradient with respect to matrix U is then given by

$$\frac{\partial J(U)}{\partial U} = \frac{1}{N}(UA - B).$$

The online dictionary learning algorithm [5] is summarized in algorithm 52.

Algorithm 52: Online dictionary learning

Input: Data $x_1, ..., x_N$, dictionary U with normalized columns and number of nonzero coefficients K, learning rate γ.

Initialize $U^{(0)} \in \mathbb{R}^{n\times m}$ and sufficient statistics $A^{(0)} = B^{(0)} = 0$.

for $n = 1,...,N$ **do**

Draw x_n from the training data set.

Get sparse code via

$$y_n = \arg\min_{y\in\mathbb{R}^m} \frac{1}{2}\left\|x_n - U^{(k)}y_n\right\|_2^2 + \lambda\left\|y_n\right\|_1$$

Update sufficient statistics

$$A^{(n)} = A^{(n-1)} + y_n y_n^T, B^{(n)} = B^{(n-1)} + x_n y_n^T.$$

Update dictionary via gradient descent:

$$U^{(n)} = U^{(n-1)} - \gamma\frac{1}{n}(U^{(n-1)}A^{(n)} - B^{(n)}).$$

end

Output: Dictionary U.

29.1.4 Non-negative matrix factorization

From SVD and dictionary learning, we decompose each data vector as linear combination of k basis vectors or 'atoms'. For example, SVD of the data matrix X yields

$$x_i = \sum_{m=1}^{k} U_m \sigma_m V_{mi}$$

where $X = U\Sigma V$ is obtained from SVD. And the order of the basis are sorted based on importance. SVD allows negative coefficients and basis vectors can cancel with each other. However, in many application, like images and text, negative elements causes difficulties in interpretation.

Non-negative matrix factorization (NMF)[6, 7] seeks to decompose the data matrix X while restricting all components to be non-negative.

Definition 29.1.1 (Non-negative matrix factorization, NMF). *[6, 7] Let X be the data matrix partitioned as $X = [x_1, x_2, ..., x_N], x_i \in \mathbb{R}^p$, the non-negative matrix factorization problem is to seek minimizers of the following optimization problem*

$$\min_{F_{ij} \geq 0, G_{ij} \geq 0} \|X - WH\|_F^2$$

where $W \in \mathbb{R}^{p \times k}, H \in \mathbb{R}^{k \times N}$.

In NMF, each data vector is decomposed as the 'additive'/positive linear combination of k basis vectors(in each column of W), given as

$$x_i = \sum_{m=1}^{k} W_m H_{mi}$$

and each column of H is the loading/coefficient for the k basis. Usually $p \gg k$ such that we can learn low dimension representations.

There are additional difference and connection between NMF and SVD.

Remark 29.1.7 (NMF vs. SVD).

- SVD requires orthogonality between basis vector, which will lead to unnatural basis vectors.
- SVD basis vectors are usually used for NMF initialization in the optimization.

NMF optimization can be carried out via gradient descent[1]. The following iterative multiplicative update method is a variant of gradient descent while maintaining non-negativeness during the process.

1 A Python implementation on NMF can be found in Nimfa.

Methodology 29.1.1 (iterative multiplicative update method for NMF). *Given initial $W, H > 0$. We can update W, H using following iterative multiplicative update rule until convergence.*

$$H_{lj} \leftarrow H_{lj} \frac{\left(W^{\mathrm{T}} X\right)_{lj}}{\left(W^{\mathrm{T}} W H\right)_{lj}}$$
$$W_{il} \leftarrow W_{il} \frac{\left(X H^{\mathrm{T}}\right)_{il}}{\left(W H H^{\mathrm{T}}\right)_{il}}$$

Remark 29.1.8 (interpret multiplicative update vs.gradient descent and convergence). Consider loss function

$$J(W, H) = \frac{1}{2}\|X - WH\|^2 = \frac{1}{2}\sum \left[X_{ij} - (WH)_{ij}\right]^2$$

The gradients are given by

$$\begin{aligned}
\frac{\partial J(W,H)}{\partial W_{il}} &= -\sum_j \left[X_{ij} - (WH)_{ij}\right] H_{lj} \\
&= -\left[\left(X H^{\mathrm{T}}\right)_{il} - \left(W H H^{\mathrm{T}}\right)_{il}\right] \\
\frac{\partial J(W,H)}{\partial H_{lj}} &= -\left[\left(W^{\mathrm{T}} X\right)_{lj} - \left(W^{\mathrm{T}} W H\right)_{lj}\right]
\end{aligned}$$

The multiplicative update rule is then given by gradient descent step

$$W_{il} = W_{il} + \lambda_{il}\left[\left(X H^{\mathrm{T}}\right)_{il} - \left(W H H^{\mathrm{T}}\right)_{il}\right]$$
$$H_{lj} = H_{lj} + \mu_{lj}\left[\left(W^{\mathrm{T}} X\right)_{lj} - \left(W^{\mathrm{T}} W H\right)_{lj}\right]$$

with

$$\lambda_{il} = \frac{W_{il}}{\left(W H H^{\mathrm{T}}\right)_{il}}, \quad \mu_{lj} = \frac{H_{lj}}{\left(W^{\mathrm{T}} W H\right)_{lj}}.$$

The multiplicative update rule guarantees convergence, at least to local minimum [7].

29.2 Advanced applications of matrix factorization methods

29.2.1 Latent sementic analysis

In text analytics like document clustering and information retrieval, we often need a convenient mathematical representation of documents. One approach, known as **vector space model (VSM)**, is to represent each document by a column vector x_i of vocabulary length L, where each component is the frequency or count of a term. The similarity between two documents i and j is obtained via cosine similarity given by

$$k(x_i, x_j) = \frac{\left\langle x_i, x_j \right\rangle}{\|x_i\|_2 \left\|x_j\right\|_2}.$$

Such model suffers, however, from its inability to cope with *synonymy* and *polysemy*. Synonymy refers to a case where two different words (say car and automobile) have the same meaning. Polysemy, on the other hand, refers to the case where a term can have difference meanings depending on the context. Consider three documents d1="aircraft, safety", d2="airplane, fast", d3="apple company, value", d4="apple, fruit, fresh". In VSM, d1 and d2 might appear dissimilar because synonymy aircraft and airplane is not captured by VSM; on the other hand, d3 and d4 might appear similar due to its inability to distinguish the two meanings of apple in Apple company and fruit apple.

Latent semantic analysis (LSA), instead, aims to construct a **latent topic space** from the **word vector space**. The similarity of documents is then measured based on their representations in the latent topic space. Documents are deemed similar if they have similar topics instead of similar terms.

In LSA, each document is represented by a column vector x_i of vocabulary length L as in VSM, where each component is the frequency of a term, typically tf-idf. We then construct a term-document matrix X by assembling multiple document vectors together. The key step in LSA is to seek a low rank approximation by performing truncated SVD (keeping top k components) on the term-document matrix X to yield

$$X \approx TSD^T,$$

where we have following interpretation:

- T s the word topic matrix with each column representing a topic. The component j in topic vector T_i represents the weight of term j in topic i. The column space of T is called topic vector space. Clearly, T is a subspace of word vector space.
- Σ is the concept strength matrix contains singular values that also reflects the importance of each topic.

- Denote $Y = SD$, then component j in Y_i is the weight of topic j in document i, where large value indicates the importance and dominance of the topic. A documents is a linear combination of topics (i.e., a document is a mixture of multiple topics), i.e., $x_i \approx \sum_{n=1}^{k} y_{in} t_i$.

For each document, y_i is its low-dimensional representation in the latent topic space. Given a new document vector x in word vector space, its topic space representation is given by

$$y = T^T x.$$

In applications, LSA offers the topic space representation of documents, which is usually advantageous compared to word vector model representation, including

- alleviate impact from synonyms and polysems,
- filtering out noise by considering only essential components of term-document matrix.
- dimension reduction that reduces storage demands.

However, SVD-based LSA has following weakness:

- impossible to interpret due to mixed signs in topic vectors
- orthogonal restriction on basis vector (i.e., columns in T) can prevent the discovery of more natural topic space.
- the truncation point k is hard to determine.

As a demonstration, we perform LSA on the 20-news-group text data. We select articles in the five categories of 'rec.baseall', 'soc.religion.christian', 'comp.graphics', 'sci.space','talk.politics.guns'. Singular value spectrum agrees with our expectation that there are only several distinct topics. The most frequent words in the top 8 topics are in Table 29.2.1. clearly, topic 2, 3, 4 are closely related to topic 'soc.religion.christian', 'talk.politics.guns', and 'comp.graphics'; however, other topics are not immediately clear to us.

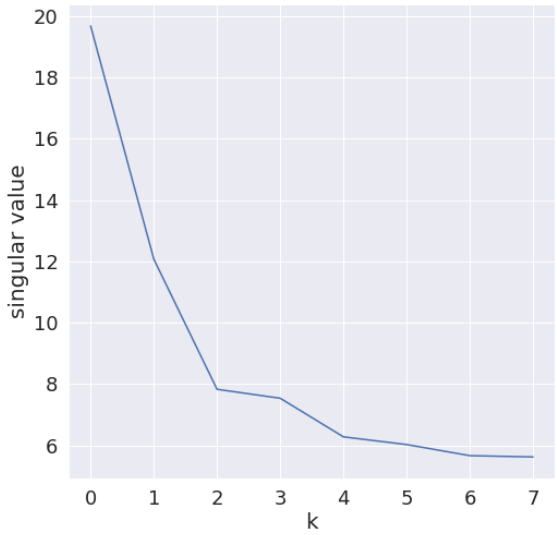


Figure 29.2.1: Singular value spectrum of LSA on 20-news-group text data.

edu	rutgers	gun	graphic	god	stratus	stratus	uiuc
rutgers	christian	stratus	comp	the	com	henry	cso
christian	athos	politics	file	graphic	digex	toronto	stratus
com	geneva	talk	image	image	access	zoo	sw
cmu	god	com	ac	nasa	sw	sw	cdt
c	religion	sw	uk	comp	cdt	cdt	uxa
apr	soc	cdt	animation	people	pat	alaska	au
athos	may	fbi	format	jesus	net	spencer	nasa
net	hedrick	people	edu	gov	rutgers	zoology	transfer
geneva	aramis	law	window	uk	transfer	utzoo	gov
news	igor	culture	sys	he	ti	transfer	ux
srv	approved	batf	bit	time	rocket	aurora	irvine
the	jesus	weapon	color	sin	prb	rocket	jbh
god	church	right	thanks	don	ibm	com	rocket
state	sin	government	picture	it	comp	nasa	god
may	bible	transfer	amiga	this	christian	nsmca	space
religion	christ	atf	ca	think	online	gov	illinois
space	christianity	firearm	au	life	graphic	space	urbana
ohio	faith	koresh	gif	ac	dseg	utnut	image
cantaloupe	he	state	ibm	but	communication	tavares	indiana

Table 29.2.1: Most frequent words in the top 8 topics

29.2.2 Collaborative filtering in recommender systems

In product (books, movies, goods, etc) recommendation applications, we are given a **rating matrix** [Table 29.2.2], which contains customer's rating on each item. This rating matrix has many missing entries, representing the customer has not yet rate and bought the item. Our goal is to predict these missing entries and recommend items with highest predicted ratings to customers.

A widely used method in recommender system is collaborative filtering. There are two types filtering:

- **User-based filtering**: the system recommend products to customers that their similar users like. For example, based on their past behavior, Alice and Tom are

identified as similar users. When Alice likes a new product, this product will be also recommended to Tom. Intuitively, customers are similar if their product rating vectors are similar according to some distance measure such as Jaccard or cosine distance.

- **Item-based filtering**: the system recommend similar products to customers based on what they recently bought or liked. For example, if Alice, Tom and Jack all gave high ratings to two movies, then these two movies are identified as similar movies; one of them will be recommended to customers who like the other.

Note that item-based filtering is similar to **content-based recommendation**, where products are identified as similar based on their product profiles like description and function, or intrinsic properties, instead of user ratings.

A matrix factorization approach to predicting the missing values in a rating matrix is to find two low-rank matrices U and V such that $R \approx UV^T$. We interpret columns in the U, V matrices as latent factors, such as the style/genre of movies [Figure 29.2.2]. We use movie recommendation as an example. Columns in V can be interpret as action movie fan, romance movie fan, etc. Each movie fan is represented by the ratings on all movies; specifically, an action movie fan will tend to give high ratings to action movies.[2] Each user's characteristics is decomposed as a linear combination of fan characteristics. For example, *Alice=0.15 × action movie fan + 0.3 × romance movie fan*. Similarly, each movie can also be decomposed into linear combination of style/genre. For example, *Titanic= 0.2 × action + 0.7 × romance*.

Then a movies' rating is represented by [8, p. 96]

$$\begin{aligned} r_{ij} &\approx \sum_{s=1}^{k} u_{is} \cdot v_{js} \\ &= \sum_{s=1}^{k} (\text{affinity of customer i to genre s}) \times (\text{affinity of movie j to genre s}) \end{aligned}$$

2 If we use SVD to decompose the rating matrix, the magnitude of singular value represents the strength of the genre.

	SW1	SW2	HP1	HP2	TW	BM
A	4				4	
B	5	5				5
C			4			5
D					5	
E		5	5	5		

Table 29.2.2: Rating matrix or utility matrix, where each row is a user's ratings for different movies. SW1, SW2 are Star wars episodes; HP1 and HP2 are Harry Potter episodes; TW is Twilight; BM is Batman.

Now we introduce a basic optimization framework to perform matrix factorization on the rating matrix.

Definition 29.2.1 (optimization problem for matrix factorization based recommender system). *Let S bet he set of all observed customer-rating pairs (i, j) in the rating matrix $R \in \mathbb{R}^{N\times M}$, in which there are N customers and M products. We seek low-rank matrix $U \in \mathbb{R}^{N\times k}, V \in \mathbb{R}^{M\times k}, k \ll \min(M, N)$, such that*

$$\min_{U,V} J = \frac{1}{2} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2,$$

Columns in U and V are called user and item factors, respectively.

Usually we use stochastic gradient descent to solve the optimization problem. Denote $e_{ij} = r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}$, the gradient respect to elements in U, V are given by

$$
\begin{aligned}
&\frac{\partial \frac{1}{2} \sum_{ij} e_{ij} e_{ij}}{\partial u_{mn}} \\
&= \sum_{ij} e_{ij} \frac{\partial e_{ij}}{\partial u_{mn}} \\
&= -\sum_{ij} e_{ij} \frac{\partial \sum_k u_{ik} v_{jk}}{\partial u_{mn}} \\
&= -\sum_{ij} e_{ij} \sum_k \delta_{mi} \delta_{kn} v_{jk} \\
&= -\sum_{ij} e_{ij} \delta_{mi} v_{jn} \\
&= -\sum_j e_{mj} v_{jn}
\end{aligned}
$$

Similarly

$$
\begin{aligned}
&\frac{\partial \frac{1}{2} \sum_{ij} e_{ij} e_{ij}}{\partial v_{mn}} \\
&= -\sum_{ij} e_{ij} \sum_k \delta_{mj} \delta_{kn} u_{ik} \\
&= -\sum_{ij} e_{ij} \delta_{mj} u_{in} \\
&= -\sum_i e_{im} u_{in}
\end{aligned}
$$

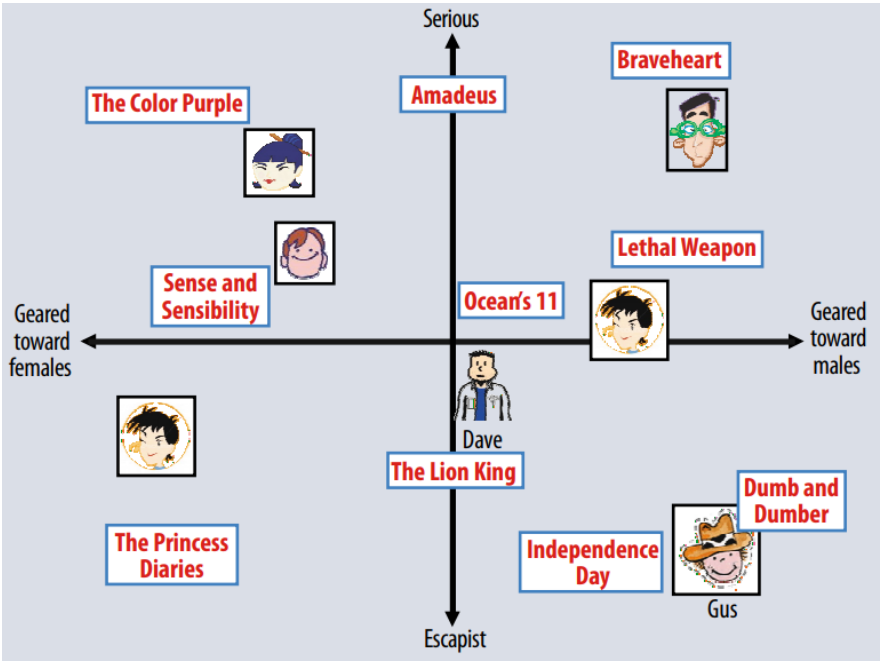


Figure 29.2.2: A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist. [9]

Algorithm 53: Stochastic gradient descent for matrix factorization based recommender systems.

Input: Rating matrix R, learning rate α

Initialize matrices U and V.

repeat

  Compute the elements in the residual matrix $E = R - UV^T$ (E will have missing entries as in R) for each filled entry index (i, j)

  simultaneously update

$$u_{ij} = u_{ij} + \alpha \sum_k e_{ik} v_{kj}$$

$$v_{ij} = v_{ij} + \alpha \sum_k e_{ki} u_{kj}$$

until *stopping criteria is met*;

Output: U, V.

Remark 29.2.1 (non-convexity and orthonormality of factors).

- This optimization problem is not convex, and thus gradient descent based method can often only find local minimum.

- Note that user factors and item factors are generally not normalized and orthogonal to each other.

Remark 29.2.2 (compact matrix form for derivatives). Let the objective function be

$$J = \frac{1}{2} Tr(EE^T),$$

where $E = (R - UV^T)$ and $e_{ij} = 0$ if rating is missing.

Then from matrix trace derivative properties [Lemma A.8.9], we have

$$\frac{\partial E}{\partial U} = EV, \frac{\partial E}{\partial V} = E^T U.$$

Remark 29.2.3 (adding biased terms and regularization). [9] For example, It is found that rating data usually exhibits large systematic tendencies for some customers to give higher ratings than others, and for some items to receive higher ratings than others. We use following item to account for the customer and item biases:

$$b_{ui} = \mu + b_i + b_c$$

where μ is the overall average rating, b_c accounts for the bias of customer c, and b_i accounts for the bias of item i.

We can also add regularization terms on the factor coefficients to prevent overfitting. The final optimization problem will be

$$\min_{U,V} \sum_{(i,j)\in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} v_{js} - \mu - b_u - b_i)^2 + \lambda(\|U\|_F^2 + \|V\|_F^2) \right)$$

29.2.3 Co-occurrence based word embedding

To efficiently carry out tasks like clustering, classification based on text data, we usually need to find numerical representation of text data such that many algorithms can process them. One naive way to represent the feature of a word as the one-hot word vector, whose length of the typical size of a dictionary. Such sparse representation does not capture the relations among words (i.e., meanings, lexical semantic) and are thus not considered as a good feature for advanced natural language processing tasks, such as language modeling. A much better alternative is to embed each word vector into a smaller dense vector (typical dimensionality ranges from 25 to 1,000) that encodes semantic meaning.

Here we introduce a way to obtain low-dimensional representation of a word vector that capture the semantic relation between words by performing SVD on a co-occurrence matrix of a large corpus. Co-occurrence matrix is a big matrix whose entry encode the frequency of a pair of words occurring together within a fixed length context window. More formally, let M be a co-occurrence matrix, and we have

$$M_{ij} = \frac{\#(w_i, w_j)/n_{pair}}{\#(w_i)/n_{words} \cdot \#(w_j)/n_{words}}$$

where $\#(w_i, w_j)$ is the number of co-occurrence of words w_i and w_j within a context window, n_{pair} is the total number pairs, n_{words} is the total number of words.

Another popular matrix to capture the co-occurrence information is the the **pointwise mutual information (PMI)**. PMI entry for a word pair is defined as the probability of their co-occurrence divided by the probabilities of them appearing individually,

$$M_{ij}^{PMI} = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)} \approx \log M_{ij}.$$

The co-occurrence information captures to some extent both semantic and syntactic information. For example, terms tend to appear together either because they have related meanings (a semantic relationship, e.g., *write* and *book*) or because the grammar rule specifies so (a syntactic relation, e.g., verbs and *to*).

By using truncated SVD to decompose the co-occurrence matrix, we obtain the low-dimensional word vectors that preserve the co-occurrence information, or the semantic and syntactic relation implied by the co-occurrence information. For example, in the low-dimensional representation, apple and pear are expected to be closer (in terms of Euclidean distance of the embedding vector) than apple and dog.

More formally, via truncated SVD, we have factorization

$$M \approx UV^T$$

where $M \in R^{N \times N}$, N is the size of the one-hot vector, $U, V \in \mathbb{R}^{N \times k}, k << N$. Columns of U are the basis vector in latent word space. Each row in V is the low dimensional representation of a word in the latent word space.

29.3 Manifold learning

29.3.1 Overview

Manifold learning is a subset of nonlinear dimensional reduction algorithms that aim to discover the low-dimensional intrinsic structures from high-dimensional data. By identifying key low-dimensional description of a high-dimensional data set, manifold learning has been used to improve modeling in diverse areas, including time-series prediction[10], automatic recommendation systems for movies, songs, and products[11, 12], modeling of protein folding and viral assembly dynamics [13–15], defect formation in colloidal system[16–18], In optimal control applications, manifold learning and its out of sample extension[19] methodology has used to provide the low-dimensional description of the state space and then learn control strategies based on low dimensional description.

In this section, we go over several manifold learning algorithms of fundamental importance. Manifold learning algorithms seek a low-dimensional projection that maximally preserves some important quantity of the data, e.g., the geodesic distances between pairs of data points (Isomap[20]). We start with the a basic tool, multidimensional scaling, since its idea and result will be repeatedly used in many manifold learning algorithms.

29.3.2 Preliminary: multidimensional scaling (MDS)

29.3.2.1 *Motivation*

Suppose we are provided a set of n objects $X \in \mathcal{X}$, and a metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that returns a distance between them. Or in some cases, we are simply given a matrix $D \in \mathbb{R}^{n \times n}$, where each entry $D_{i,j}$ is the distance between the i th and j th point. Multi-Dimensional Scaling (MDS) has the goal of taking such a distance matrix D for n points and giving low-dimensional (typically) Euclidean coordinates y_i of these points such that

$$D_{ij} \approx \left\| y_i - y_j \right\| .$$

The intuition is that if we had some original data set A that can be used to compute D, we could just apply PCA on AA^T to find the embedding. Since in the MDS context, we are only given D, we can derive a matrix from D that will act like AA^T and then apply PCA to it.

We start with the characterization of distance between points. The distance matrix can be either constructed from Euclidean distance metric or other metrics.

Example 29.3.1 (distance matrix associated with a set of points). Consider a set of N points $x_1, x_2, ..., x_N \in \mathbb{R}^D$.

- The **Euclidean distance matrix** $D \in \mathbb{R}^{N \times N}$ associated with the data $\{x_i\}$ is defined by
$$D_{ij} = \left\| x_i - x_j \right\| = \sqrt{x_i^T x_i - 2x_i^T x_j + x_j^T x_j}.$$
- The **L1 distance matrix** $D \in \mathbb{R}^{N \times N}$ defined by
$$D_{ij} = \|x_1 - x_2\|_1 .$$

Now the multidimensional scaling problem can be summarized by the following definition.

Definition 29.3.1 (multidimensional scaling problem). *Consider a set of N points $x_1, x_2, ..., x_N \in \mathbb{R}^D$ and its associated distance matrix $D \in \mathbb{R}^{N \times N}$. The goal of the multidimensional scaling problem is to find a set of low-dimensional representation of $y_1, ..., y_N, y_i \in \mathbb{R}^d, d \ll D$ for $z_1, ..., z_N$ such that the distance matrix is maximally preserved:*
$$\min_Y \|D_Y - D\|_F^2$$
where D_Y is the distance matrix from lower dimensional representation Y.

29.3.2.2 *Solution to classical MDS*

The MDS problem does not have analytical solutions and requires iterative method to solve it. Note that SVD can be used to seek low dimensional representations that approximate inner product matrix [Theorem 14.2.6]. One common way to solve MDS problem to first assume distance matrix D is generated by some high dimensional representations x via the Euclidean distance metric, and then to convert distance matrix matrix to inner product matrix. Finally, we seek low-dimensional representations that minimize the reconstruction error of the inner product matrix. We first examine the relationship between an inner product matrix and a distance matrix.

Lemma 29.3.1 (The distance matrix to inner product conversion operator). *Consider a distance matrix $D \in \mathbb{R}^{N \times N}$ where $D_{ij} = \left\| x_i - x_j \right\| = \sqrt{x_i^T x_i - 2x_i^T x_j + x_j^T x_j}$.*

- *The **centered inner product matrix** $B_{ij} = (x_i - \overline{x})^T (x_j - \overline{x})$ is connected to the distance matrix via*
$$B = -\frac{1}{2} HSH$$

where $H = (I - \frac{1}{N}J)$, [a] *or $H_{ij} = \delta_{ij} - 1/N, S_{ij} = D_{ij}^2$. We introduce the operator τ as*

$$\tau(D) = -\frac{1}{2}HSH.$$

- *On the other hand, if $x_1, ..., x_N$ are centered, i.e., $\sum_{i=1}^N x_i = \mathbf{0}$, we can recover D from B via*

$$D_{ij}^2 = B_{ii} - 2B_{ij} + B_{jj}^2.$$

[a] Note that H is a orthogonal projector with trace $N - 1$, thus not invertible.

Proof. (1) It can be showed that

$$D_{ij}^2 = x_i^2 + x_j^2 - 2x_i x_j$$

$$\frac{1}{N}\sum_i^n D_{ij}^2 = \sum_i x_i^2/N + x_j^2 - 2x_j\overline{x}$$

$$\frac{1}{N}\sum_j^n D_{ij}^2 = x_i^2 + \sum_j x_j^2/N - 2x_i\overline{x}$$

$$\frac{1}{N^2}\sum_i^n\sum_j^n D_{ij}^2 = \sum_i x_i^2/N + \sum_j x_j^2/N - 2\overline{x}\overline{x}$$

The centered inner product Y_{ij} has

$$B_{ij} = x_i x_j - x_j\overline{x} - x_i\overline{x} + \overline{x}\overline{x}$$

then we have

$$B_{ij} = -\frac{1}{2}(D_{ij}^2 - \frac{1}{N}\sum_j^n D_{ij}^2 - \frac{1}{N}\sum_i^N D_{ij}^2 + \frac{1}{N^2}\sum_i^N\sum_j^N D_{ij}^2)$$

That is,

$$B = \frac{1}{2}(I - \frac{1}{N}J)S(I - \frac{1}{N}J).$$

(2) Straight forward. □

Because centered inner product matrix and distance matrix can be converted to each other, in MSD, we can instead seek centered low-dimensional representation $y_1, ..., y_N \in \mathbb{R}^d$ to maximally preserving the inner product matrix as the alternative strategy to maximally preserving the distance matrix. It turns out that working with inner product matrix is much easier.

Definition 29.3.2 (MDS in alternative inner product matrix form). *Given a distance matrix/dissimilarity measure matrix $D \in \mathbb{R}^{N\times N}$, the goal of MDS is to find a centered matrix $Y = [y_1^T; y_2^T; ...; y_N^T], y_i \in \mathbb{R}^d, Y \in \mathbb{R}^{N\times d}$ (y_i is usually lower dimensional) to maximally preserving the centered inner product matrix of D, denoted by $\tau(D)$:*

$$\min_Y \left\| YY^T - \tau(D) \right\|_F^2.$$

Remark 29.3.1 (non-uniqueness of solutions). Suppose we have found the matrix Y that solves $\min_Y \left\| YY^T - \tau(D) \right\|_F^2$. If we rotate Y to yield $Y_R = YR^T$, Y' is also the solution since $Y_R Y_R^T = YY^T$.

Theorem 29.3.1 (Solution to MDS via SVD). *Let rank d approximation to $\tau(D)$ be $\tau(D) \approx U\Lambda U^T$, $U \in \mathbb{R}^{N\times N}$ via SVD [Theorem 4.9.4].*

Given a MDS problem in inner product matrix form, the optimal solution is $Y = \Lambda^{1/2}U$, or $y_{ij} = \sqrt{\lambda_j}U_{ij}, i = 1, 2, ...N, j = 1, 2..., d.$ where . Explicitly,

$$y_i = \begin{bmatrix} \sqrt{\lambda_1}U_{i1} \\ \sqrt{\lambda_2}U_{i2} \\ \vdots \\ \sqrt{\lambda_d}U_{id} \end{bmatrix}$$

Proof. Clearly, when we take $Y = \Lambda^{1/2}U$, then $YY^T = U\Lambda U^T$ is the rank d approximation to $\tau(D)$. □

Example 29.3.2 (embeddings for triangles and tetrahydron). Consider the distance matrix associated with a 2D triangle and a 3D tetrahydron, which are given by

$$D_1 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, D_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

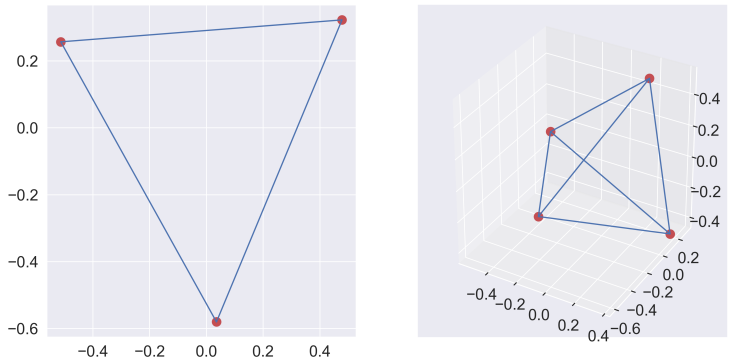


Figure 29.3.1: Triangle and Tetrahydron reconstructed from distance matrix by MDS method.

Using MDS, we can reconstruct their configurations up to translation and rotation [Figure 29.3.1].

Remark 29.3.2 (Euclidean MDS cannot capture nonlinear structure)**.**

- If we use Euclidean distance in the MDS, large distances and small distances are weighted equally. Since large distances between data points provide little information on the global structure of the data set compared to small distances between neighboring data points. Euclidean distance metric usually fails to reveal the nonlinear structure of the data [Figure 29.3.2].
- If we use different distance measure, we can make MDS for nonlinear structure; For example, in Isomap, we use geodesic distance between data points as distance measure [20].

29.3.3 Isomap

Isomap [20] can be viewed as an application of MDS on a distance metrics that is based on geodesic distance between data points. Let $X_1, ..., X_N$ be the high dimensional data set. In Isomap, we construct the distance matrix between all pairs of data via the following steps.

- We construct an adjacency weighted graph $G = (V, E)$ on the data $\mathcal{X}$, where nodes are data points and the edges connecting each node to its K nearest neighbors. For each data point, we find its nearest neighbors based on some distance measure $d_X(\cdot, \cdot)$ best approximating the local distance. The measured distances are the weights associated with the edges.

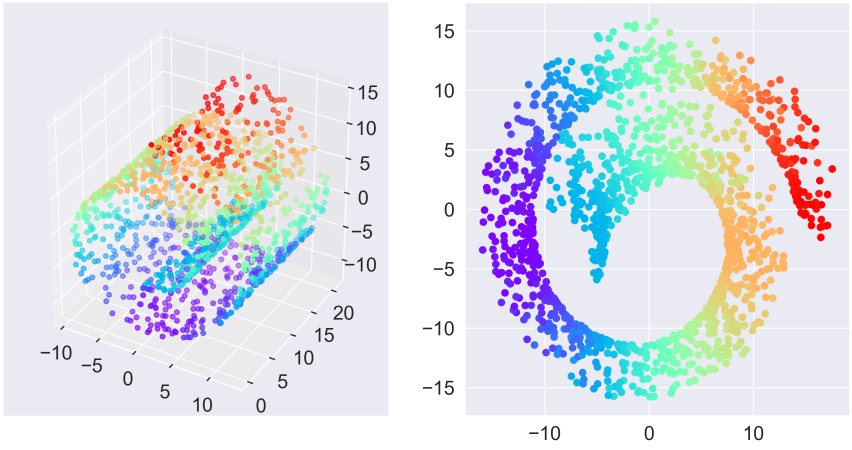


Figure 29.3.2: Application of MDS, based on Euclidean distance, to Swiss Roll data set cannot fully reveal of the global structure.

- We estimate the geodesic distance between all pairs of data points on the manifold by calculating the shortest path distances on pairs of nodes on the weighted graph. We store the geodesic distances in a matrix D_G.

With the distance matrix D_G, we spply multi-dimensional scaling (MDS) [subsection 29.3.2] to the distance matrix D_G and obtain the low-dimensional embedding $Y_1, ..., Y_N \in \mathrm{R}^K$ via the following procedures.

- Convert the distance matrix to an inner product matrix via $\tau(D_G) = -\frac{1}{2} J D_G J$, where $J = I - \frac{1}{N} 11'$ is the centering matrix.
- Perform eigendecomposition on $\tau(D_G)$ and obtain the first largest K eigenvalues $\lambda_1, ..., \lambda_K$ and their associated eigenvectors $e_1, ..., e_K$.
- The low-dimensional embedding Y_i is given by $Y_i = (\sqrt{\lambda_1} e_1^i, ..., \sqrt{\lambda_K} e_K^i)$, where e_p^i is the ith component of eigenvector e_p.

We summarize the Isomap algorithm in the following.

Algorithm 54: Isomap algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$.

1 Construct a distance matrix D_G based on geodesic distance (shortest path distance in graphs).

2 Solve the Isomap cost function minimization problem

$$\min_Y \left\| \tau(D_G) - \tau(D_Y) \right\|_F^2$$

using eigendecomposition.

Output: the low dimensional coordinates $y_1, ..., y_N \in \mathbb{R}^K$.

29.3.4 Kernel PCA

In kernel PCA, we are given a a semi-positive symmetric kernel $k(x, y)$ that measures the similarity between data i and data j via $K_{ij} = k(x_i, x_j)$. By viewing the kernel matrix as the proxy of the inner product matrix, We can then apply MDS to the kernel matrix to obtain low dimensional embeddings [algorithm 55].

There are several popular choices for the nonlinear kernel functions, such as the polynomial kernel and the Gaussian kernel, respectively

$$k_P(x, y) = (x^T y)^n \quad \text{and} \quad k_G(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2} \right).$$

PCA is a particular example of kernel PCA with the kernel being the inner product $k(x,y) = x^T y$. Note that eigendecompostion always exist since K is a symmetric positive definite matrix.

Algorithm 55: Kernel PCA algorithm

Input: Data set consists of $x_1, x_2, ..., x_N, x_i \in \mathbb{R}^D$.

1 Define a semi-positive symmetric kernel $k(x,y)$ and construct a kernel matrix K such that $K_{ij} = k(x_i, x_j)$ and **center** K;

2 Compute the eigenvectors u^i and eigenvalues λ_i centered kernel matrix K such that

$$Ku^i = \lambda_i u^i.$$

3 The low dimensional coordinate $y_i \in \mathbb{R}^p, D \gg p$ is given as

$$y_i = \begin{pmatrix} u_i^1 \sqrt{\lambda_1} \\ u_i^2 \sqrt{\lambda_2} \\ \cdots \\ u_i^p \sqrt{\lambda_2} \end{pmatrix}$$

Output: The low dimensional coordinate $y_i \in \mathbb{R}^p, i = 1, 2, ..., N$.

Lemma 29.3.2 (best low rank approximation to kernel matrix). *Let $M_{ij} = \left\langle y_i, y_j \right\rangle$, with y_i defined in algorithm 55, then M is the optimal low rank approximation to the kernel matrix, given as*

$$\min_{rank(M)=p} \|K - M\|_F^2$$

More compactly

$$M = Y^T Y, Y = [y_1, y_2, ..., y_N]$$

and M is the approximate kernel matrix constructed from low dimensional coordinates.

Proof. Directly from the low rank approximation property of SVD [Theorem 29.1.4]. Moreover,

$$M_{ij} = \left\langle y_i, y_j \right\rangle = \sum_{n=1}^{p} \sqrt{\lambda_n} u_i^n \sqrt{\lambda_n} u_j^n = \sum_{n=1}^{p} y_{in} y_{jn}.$$

□

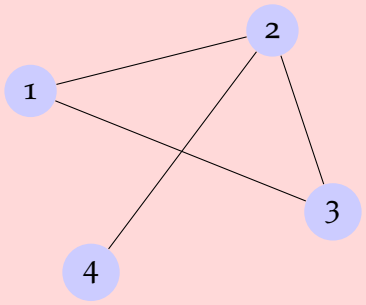
29.3.5 Laplacian eigenmap

29.3.5.1 *Preliminary: graph Laplacian*

Definition 29.3.3 (Laplacian matrix). *Let $W \in \mathbb{R}_+^{N\times N}$ be a non-negative symmetric matrix and let $D \in \mathbb{R}^{N\times N}$ be a diagonal matrix with $d_{jj} = \sum_{i=1}^{N} w_{ij}$, then the symmetric matrix $L = D - W$ is called **Laplacian matrix**.*

Example 29.3.3. A typical example of W is the adjacency matrix of an undirected graph. For example, the weight matrix represents the graph below.

$$W = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$



Lemma 29.3.3 (elementary property of Laplacian matrix). *[3, p. 140][21]Let L be a Laplacian matrix, then*

- *L is positive semi-definite.*
- *The vector of all ones $\mathbf{1}$ is the eigenvector associated with zero eigenvalue, that is $L\mathbf{1} = 0$*
- *L is diagonalizable but L is singular matrix.*
- *L has n non-negative, real-valued eigenvalues*

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

Proof. (1) Let $y \in \mathbb{R}^N$ be a nonzero vector, we have

$$y^T D y = \sum_i y^i (y_i \sum_j w_{ji}) = \sum_i \sum_j y_i^2 w_{ji} = \sum_i \sum_j y_j^2 w_{ij}$$

$$y^T W y = \sum_i \sum_j y_i y_j w_{ij}$$

where we use the fact that W is symmetric. Then we have

$$y^T (D - W) y = y^T L y = \frac{1}{2} \sum_i \sum_j w_{ij} (y_i - y_j)^2 \geq 0.$$

We can also directly prove (1) using Theorem 4.13.8 and Theorem 4.13.6.

(2) Directly from the definition of $L = D - W$. (3) L is real-valued symmetric and therefore diagonalizable. L is singular since it has eigenvalue 0. (4) From (1), we can see the L is a positive semi-definite matrix, therefore its eigenvalues are non-negative. □

Lemma 29.3.4 (number of connected components from spectrum of L). *[21] Let G be an undirected graph with non-negative weights with Laplacian L. Then multiplicity k of eigenvalue 0 of L equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace associated eigenvalue 0 has dimensionality k spanned by the indicator vectors $1_{A_1}, ..., 1_{A_k}$ of these components.*

Proof. It is easy to show that $1_{A_1}, ..., 1_{A_k}$ satisfy $L 1_{A_i} = 0$. Note that from Theorem 4.12.2, we note that the geometric multiplicity is the same as the algebraic multiplicity. Therefore, $1_{A_1}, ..., 1_{A_k}$ will span the eigenspace. □

Definition 29.3.4 (normalized graph Laplacian). *[21] Given a Laplacian L associated with a graph G, we can define **normalized Laplacian** as:*

- $L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}$
- $L_{rw} = D^{-1} L = I - D^{-1} W$.

where $D = diag(d_1, ..., d_N), d_i = \sum_{ij} w_j$.

Theorem 29.3.2 (spectrum properties of normalized Laplacian). *The normalized Laplacian satisfy the following properties:*

- *For every $v \in \mathbb{R}^N$, we have*

$$v^T L_{sym} v = \frac{1}{2} \sum_{i,j} w_{ij} (\frac{v_i}{\sqrt{d_i}} - \frac{v_j}{\sqrt{d_j}}).$$

- *λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2}u$.*
- *λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ and u satisfy*

$$Lu = \lambda Du.$$

- *o is an eigenvalue L_{rw} with has the eigenvector of the constant one vector 1. o is an eigenvalue of L_{sym} with eigenvector $D^{1/2}1$.*
- *L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 \leq \lambda_1 \leq ... \leq \lambda_n$.*

Proof. Straight forward from definitions of normalized graph Laplacian. □

Remark 29.3.3 (short summary).

- L_{sym} and L_{rw} have exactly the same eigenvalues.
- L has different nonzero eigenvalues from L_{sym} and L_{rw}. However, they have the same number of zero eigenvalues.

Corollary 29.3.2.1 (number of connected components from spectrum of L). *[21] Let G be an undirected graph with non-negative weights with Laplacian L. Then multiplicity k of eigenvalue o of L, L_{sym}, L_{rw} equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace associated eigenvalue o of L, L_{rw} has dimensionality k spanned by the indicator vectors $1_{A_1}, ..., 1_{A_k}$ of these components. The eigenspace associated eigenvalue o of L_{sym} has dimensionality k spanned by the indicator vectors $D^{1/2}1_{A_1}, ..., D^{1/2}1_{A_k}$ of these components.*

29.3.5.2 *Laplacian eigenmap*

Let $x_1, ..., x_N$ be the data lying on a high dimensional manifold M. Let $W \in \mathbb{R}^{N \times N}$ be the matrix such that W_{ij} characterized the affinity/similarity of between the data x. The Laplacian eigenmap problem is to find a **low dimensional embedding** $Y = [y_1, ..., y_N], y_i \in \mathbb{R}^D$, such that if x_i and x_j are closed to each other, then so are y_i and y_j. The goal can be formulated as

$$\min_{y_i, ..., y_N} \sum_{i,j} w_{ij} \left\| y_i - y_j \right\|^2.$$

where a larger weight $w_{ij} \geq 0$ indicates that the two data points i and j are "similar", and y_i and y_j need to be close in the low-dimensional space to minimize the objective function.

One common choice of weight that preserves similarity is based on local distance and K nearest neighbors (KNN), given by

$$w_{ij} = \begin{cases} e^{-\frac{d\left(x_i \cdot x_j\right)^2}{2\sigma^2}} & \text{if } x_i, x_j \textit{ are} \text{ KNN to each other} \\ 0 & \text{otherwise} \end{cases}$$

Here we use negative exponential to penalize large distances.

If we expand $\phi(Y) = \sum_{i,j} W_{ij} \left\| y_i - y_j \right\|^2$,, we can connect $\phi(Y)$ with the graph Laplacian (similar derivation in Lemma 29.3.3]

$$\begin{aligned} \phi(Y) &= \sum_{ij} w_{ij} \left(\left\| y_i \right\|^2 + \left\| y_j \right\|^2 - 2 y_i^\top y_j \right) \\ &= 2 \sum_j d_{jj} y_j^\top y_j - 2 \sum_{ij} w_{ij} y_i^\top y_j \\ &= 2Tr\left(YDY^T\right) - 2Tr\left(YWY^T\right) = 2Tr\left(YLY^T\right) \end{aligned}$$

where D is a diagonal matrix with $D_{jj} = \sum_i w_{ij}$.

Further, we notice that there are several trivial solutions to the problem $\min_Y \phi(Y)$.

- The first trivial solution is is $Y = 0$, where low dimensional representations are just the origin.
- Another trivial solution is all y_i are chosen the same element.

To prevent these trivial solutions, Laplacian eigenmap requires the low-dimensional representation Y to satisfy the following additional constraints[3, p. 136]:

$$YD\mathbf{1} = \mathbf{0} \quad \text{and} \quad YDY^T = I.$$

Now the Laplacian eigenmap problem is given by

$$\min_Y Tr\left(YLY^T\right) \quad \text{s.t.} \quad YD\mathbf{1} = \mathbf{0} \quad \text{and} \quad YDY^T = I.$$

The solution of this optimization problem is given by

Theorem 29.3.3 (solution to Laplacian eigenmap problem). *[3, p. 136] The solution to the optimization problem*

$$\min_Y Tr\left(YLY^T\right) \quad \textit{s.t.} \quad YD\mathbf{1} = \mathbf{0} \quad \textit{and} \quad YDY^T = I$$

the second to top $(d+1)$ smallest generalized eigenvalues and eigenvectors of $Lu = \lambda Du$.

Proof. Notice that the Lagrangian function for this problem can be written as

$$\mathscr{L}(Y, \lambda, \Lambda) = Tr\left(YLY^T\right) + \gamma^\top YD\mathbf{1} + Tr\left(\Lambda\left(I - YDY^T\right)\right)$$

where $\beta \in \mathbb{R}^d$ and $\Lambda = \Lambda^\top \in \mathbb{R}^{d\times d}$ are, respectively, a vector and matrix of Lagrange multipliers. Computing the derivative of $\mathscr{L}$ with respect to Y and setting it to zero yields $2YL + \beta\mathbf{1}^\top\mathcal{D} - 2\Lambda Y\mathcal{D} = \mathbf{0}$. Multiplying on the right by $\mathbf{1}$ and using the constraints $L1 = 0$ and $YD1 = 0$, we obtain $\lambda = 0$. As a consequence,

$$YL = \Lambda Y\mathcal{D} \implies LY^\top = DY^T\Lambda$$

Because $YLY^T = \Lambda YDY^T$, to minimize the objective function, we need to choose the eigenvectors associated with the smallest eigenvalues. □

Algorithm 56: Laplacian Eigenmap algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$ on a manifold M

1 Find the K nearest neighbors of each data point $x_i, i = 1, ..., N$ according to some distance function $d(x_i, x_j)$ defined on M.

2 Construct similarity matrix W, where

$$W_{ij} = \begin{cases} e^{-d(x_i,x_j)^2/\sigma^2}, \ if \ i,j \ are \ K \ nearest \ neighbors \\ 0, \ otherwise \end{cases}$$

3 Construct the graph Laplacian $L = D - W$.

4 Solve the second to top $(d+1)$ smallest generalized eigenvalues and eigenvectors of $Lu = \lambda Du$.

5 Assign each data point i with the new lower dimensional coordinate $y_i = (u_{1,i}, u_{2,i}, ..., u_{d,i})$.

Output: The low dimensional coordinates $y_i \in \mathbb{R}^d, i = 1, ..., N$

Remark 29.3.4 (We ignore eigenvectors associated with zero eigenvalue).

- Note that in Laplacian eigenmap, we will ignore the top eigenvector, because the top eigenvector (the multiplicity is 1 usually)is constant 1 vector that does not provide extra information.

29.3.6 Diffusion map

Diffusion map is a manifold learning technique based on Markov chain transition dynamics on the high-dimensional data. Diffusion map construct a Markov chain transition matrix based on the distance metrics, particularly distance among neighboring points, of high-dimensional data points. Usually, this is achieved by define a semi-positive symmetric kernel $k(x, y)$ and construct a distance matrix K such that $K_{ij} = k(x_i, x_j)$. The most commonly used kernel is the Gaussian kernel given by

$$k_G(x, y) = \exp\left(-\frac{\|x - y\|^2}{\sigma^2}\right),$$

which penalizes large distances.

The distance between neighboring points are then converted to transition probabilities, which enables the calculation of distance, also called **diffusion distance**, between arbitrary two data points in the context of Markov chain dynamics. More formally, we have definition.

Definition 29.3.5 (diffusion distance). *In a Markov chain P satisfying detailed balance, we can define the diffusion distance on a time step n between different state i and j as*

$$D(i, j) = \left\|P(i, \cdot) - P(j, \cdot)\right\|_{1/\pi}^2 = [\sum_{k=1}^{n} \frac{(P_{i,k} - P_{j,k})^2}{\pi_k}]^{0.5}$$

Remark 29.3.5 (interpretation).

- The distance between nodes i, j is given by starting the Markov chain at each node i, j let them evolve after a fixed time step and computing the mean square distance between the two resulting distributions.
- The distance computation is weighted by the inverse equilibrium density $1/\pi$.

By performing eigendecomposition on the transition matrix, we can obtain low-dimensional representation of original data points that maximally preserve diffusion distance between points.

The feasibility of performing eigendecomposition is given by the following Lemma.

Lemma 29.3.5 (eigendecomposition of transition matrix). *The transition matrix M has the following properties*

- *M can be transformed symmetric matrix via $V = \Pi M \Pi^{-1}$ where Π is the diagonal matrix with entries $\sqrt{\pi_1}, ..., \sqrt{\pi_n} (\pi = (\pi_1, ..., \pi_n)$ is the equilibrium probability vector).*

- *Let $\lambda_1, ..., \lambda_n$ and $w_1, ..., w_n$ be the **real eigenvalues** and unit eigenvectors of V. Then M has the **same real eigenvalues**. The left eigenvectors of P are given as*

$$\psi_j = \Pi w_j$$

The right eigenvectors of P are given as

$$\phi_j = \Pi^{-1} w_j$$

- *M has the following spectral decomposition:*

$$M = \sum_{k=1}^{n} \lambda_k \phi_k \psi_k^T = \sum_{k=1}^{n} \lambda_k \Pi^2 \phi_k \phi_k^T$$

- *Except for the first eigenvalue having value 1, then all other eigenvalues with their absolute value strictly less than 1.*

Proof. Directly from Theorem 20.5.1. □

The representation of each data point x_i in the diffusion space is given as

$$\Phi(i) = \begin{bmatrix} \lambda\phi_1(i) \\ \lambda\phi_2(i) \\ \lambda\phi_3(i) \\ \cdots \end{bmatrix}$$

It can be showed in the following Lemma that such low representation maximally preserves the diffusion distnace.

Lemma 29.3.6 (preserving diffusion distance). *[22][23] The diffusion distance $D(i,j)$ on a time step n between different state i and j is given as*

$$[D(i,j)]^2 = \sum_{k=1}^{n} \lambda_k^2 (\phi_k(i) - \phi_k(j))^2$$

where ϕ_k is the kth right eigenvector of P. In other words, the representation in Euclidean space

$$\Phi(i) = \begin{bmatrix} \lambda_1\phi_1(i) \\ \lambda_2\phi_2(i) \\ \lambda_3\phi_3(i) \\ \cdots \end{bmatrix}$$

preserves the diffusion distance.

Proof. Note that $P_{i,j} = \sum_{k=1}^{n} \lambda_k \phi_k(i)\phi_k(j)\pi_k$. Then

$$D(k,m)^2 = (\sum_i \sum_j \lambda_i \phi_i(k)\phi_i(j) - \sum_i \sum_j \lambda_i \phi_i(m)\phi_i(j))^2$$

where orthonormality is needed to prove. □

The complete algorithm is summarized in the following.

Algorithm 57: Diffusion map algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$.

1 Define a semi-positive symmetric kernel $k(x, y)$ and construct a kernel matrix K such that $K_{ij} = k(x_i, x_j)$;

2 Construct the diagonal matrix D with $D_{jj} = \sum_{i=1}^{N} K_{ij}$

3 Normalize each row of K, given as $M = D^{-1}K$;(now M has the row sum 1, which is also called stochastic matrix.)

4 Compute the second to $k+1$ largest eigenvalues and eigenvectors of M)

5 The representation of each data point x_i is the diffusion space is given as

$$\Phi(i) = \begin{bmatrix} \lambda\phi_1(i) \\ \lambda\phi_2(i) \\ \lambda\phi_3(i) \\ \cdots \end{bmatrix}$$

Output: the coordinate in the diffusion space.

Remark 29.3.6 (We ignore eigenvectors associated with zero eigenvalue).

- Note that in diffusion mapping, we will ignore the top eigenvector, because the top eigenvector (the multiplicity is 1 usually)is constant 1 vector that does not provide extra information.

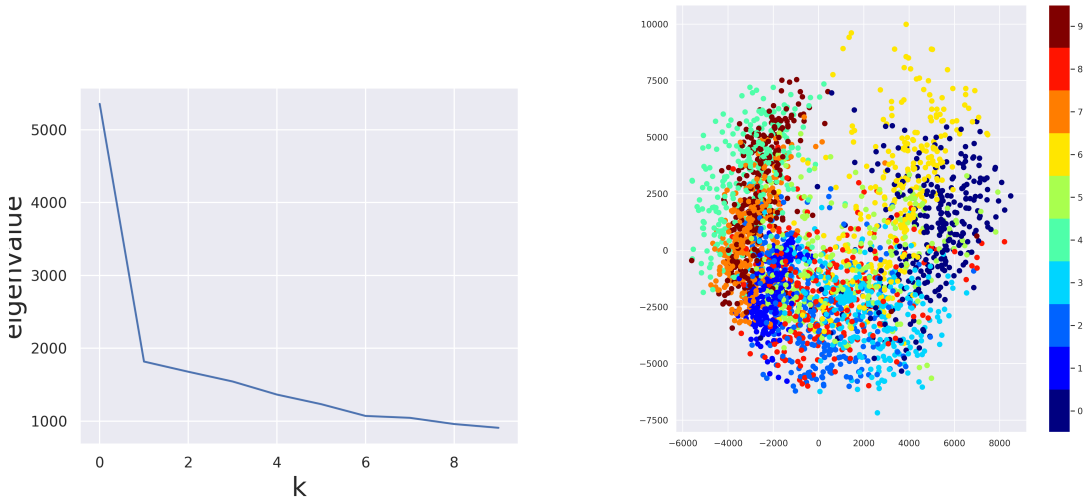
Remark 29.3.7 (out of sample extension). For out-of-sample extensions, see [24][25].

29.3.7 Application examples

29.3.7.1 *MNIST*

Here we use Isomap to find the low dimension embedding of the MNIST data set [Figure 29.3.3]. The eigenvalue spectrum [Figure 29.3.3(a)] of the distance matrix suggest that two dimensional embedding can capture the majority of distance between the original raw data set. In Figure 29.3.3(b), we can see that same digits are mostly clustered together, although there is also significant overlaps.

We can further apply Isomap to find the low-dimensional embedding of the same digit. The 2D embedding in Figure 29.3.4 suggest that the low-dimensional manifold in which these sample lie correspond to the variation of the pose.



(a) Eigenvalues of shortest path distance matrix in Isomap analysis of MNIST dataset.

(b) Two-dimensional embedding of MNIST dataset via Isomap analysis.

Figure 29.3.3: Isomap analysis of MNIST dataset.

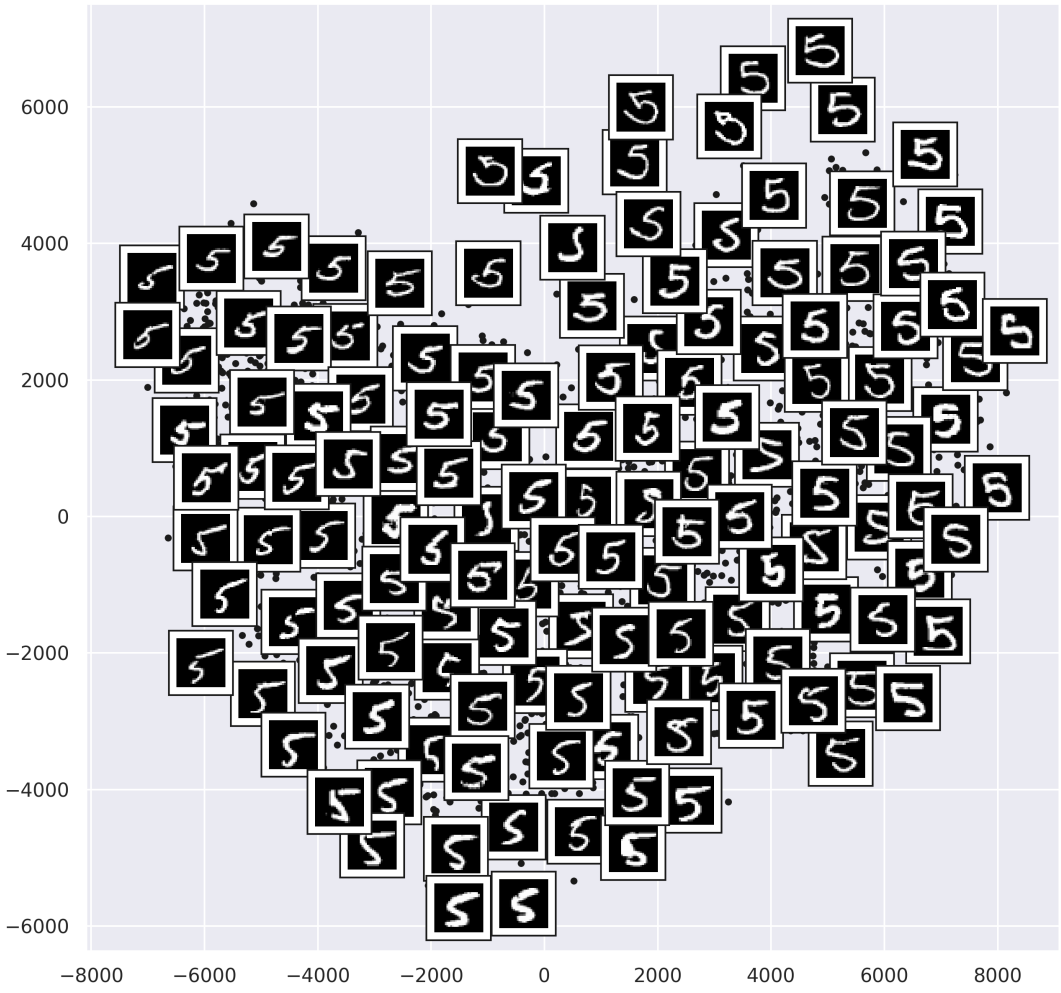


Figure 29.3.4: Isomap analysis of digit '5' in MNIST dataset

29.4 Clustering

29.4.1 Overview

Clustering is the process of clustering a collections of objects into different groups based on their similarities and differences. The similarity or differences are characterized by Euclidean distance, in the most common case, and other metrics specific to particular applications. With suitable metrics, clustering can be applied to a wide variety of objects, from points in the $\mathbb{R}^n$ to images to texts.

We will first introduce the K-means algorithms, which is the most basic and widely used clustering algorithm. Following the discussion on the potential pitfalls of the K-means algorithms, we introduce other clustering algorithm variants.

29.4.2 K-means

29.4.2.1 *Canonical K-means*

K-means clustering aims to partition n observations into k clusters in which each observation is closest to its cluster center [Figure 29.4.1]. In $\mathbb{R}^p$ with Euclidean distance metric, the resulting partition is just Voronoi diagram based on the cluster centers.

Mathematically, given a set of N points $x_1, ..., x_N \in \mathbb{R}^p$, the K-means clustering can be viewed as an optimization problem that seeks K clusters to solve

$$\arg\min_{Z,A} \sum_{i=1}^{N} \left\| x_i - z_{A(x_i)} \right\|_2^2,$$

where $Z = \{z_1, z_2, ..., z_K \in \mathbb{R}^p\}$ are the means of the K clusters, A is assignment function $A : \mathbb{R}^p \to \{1, 2, ..., K\}$ such that $A(x_i)$ assigns point i exclusively to one of the K clusters. Let S_i denote the index set $S_i = \{j : A(x_j) = i\}$. Then

$$z_j = \frac{\sum_{j \in S_i} x_j}{|S_j|}.$$

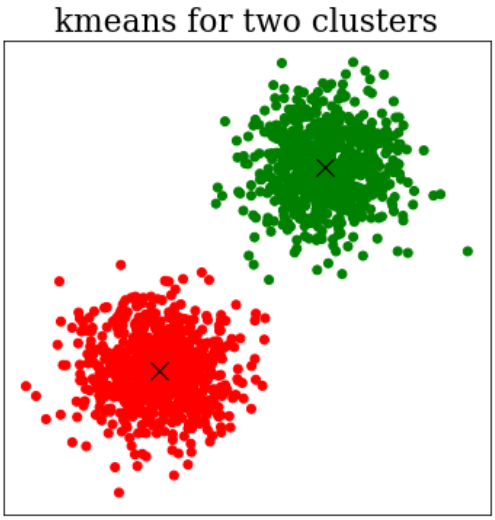


Figure 29.4.1: Demonstration of k-means clustering on a data set with two blobs.

Seeking the global optimal solution for the K mean problem is usually intractable. The most commonly used algorithm [algorithm 58] is an iterative procedure repeating two steps, starting from initially-guessed cluster centers:

- Updating assignment, where each point is assigned to newly updated clusters.
- Updating cluster centers, where cluster is updated to a new mean.

As we will show in subsubsection 29.4.5.1, the K-mean algorithm can be viewed as a variant of expectation-maximization algorithm. Therefore, the objective function during minimization process will always decrease.

The convergence criterion is either iteration approaches maximum iteration limit or the decrement is smaller than a specified tolerance. Usually, the iteration converges to local minimum; global convergence is not guaranteed.

Algorithm 58: K-means algorithm.

Input: Data set consists of $x_1, x_2, ..., x_N$. Number of clusters K.

1 Set $n = 0$, and set initial cluster center $z_1^0, ..., z_K^0$.

2 **repeat**

3 **Assignment step**: fix $z_1^n, ..., z_K^n$ and update the assignment function

$$A^n(x_i) = \arg \min_{j \in \{1,...,K\}} \|x_i - z_i\|_2^2,$$

and index set $S_i^n = \{j : A^n(x_j) = i\}$.

4 **Cluster center update step**: Given a fixed A^n, update cluster centers Z as

$$z_j^{n+1} = \frac{\sum_{j \in S_i} x_j}{|S_i^n|}, j = 1, ..., K.$$

5 set $n = n + 1$.

6 **until** *convergence condition satisfied*;

Output: Cluster center Z and index set S_i.

The canonical K-means algorithm is widely used because of its speed and simplicity. It is relatively efficient, with a computational cost given by $O(tknd)$, where t is number of iterations, k is the number of clusters, n is the number of point, and d is the dimensionality. The algorithm usually yields the best result when data points have clear clustering patterns and clusters are well separated from each other.

We also need to be aware of a number of drawbacks:

- The number of cluster centers has to be specified by the user, instead of directly learned from data. Determining the number of clusters is non-trivial, particularly for high dimensional data.
- K-means algorithm works well for spherical-shaped clusters, but not for non-spherical clusters. For the same reason, K-means algorithm is not invariant to non-linear transformations that could change the shape and geometry of the data.
- Euclidean distance measure is used in K-means algorithms, which can be unsuitable for some applications.
- Clustering results can be highly dependent on the initial cluster centers.
- Clustering results are highly sensitive to noisy data and outliers.

Some of these weaknesses are summarized in Figure 29.4.2.

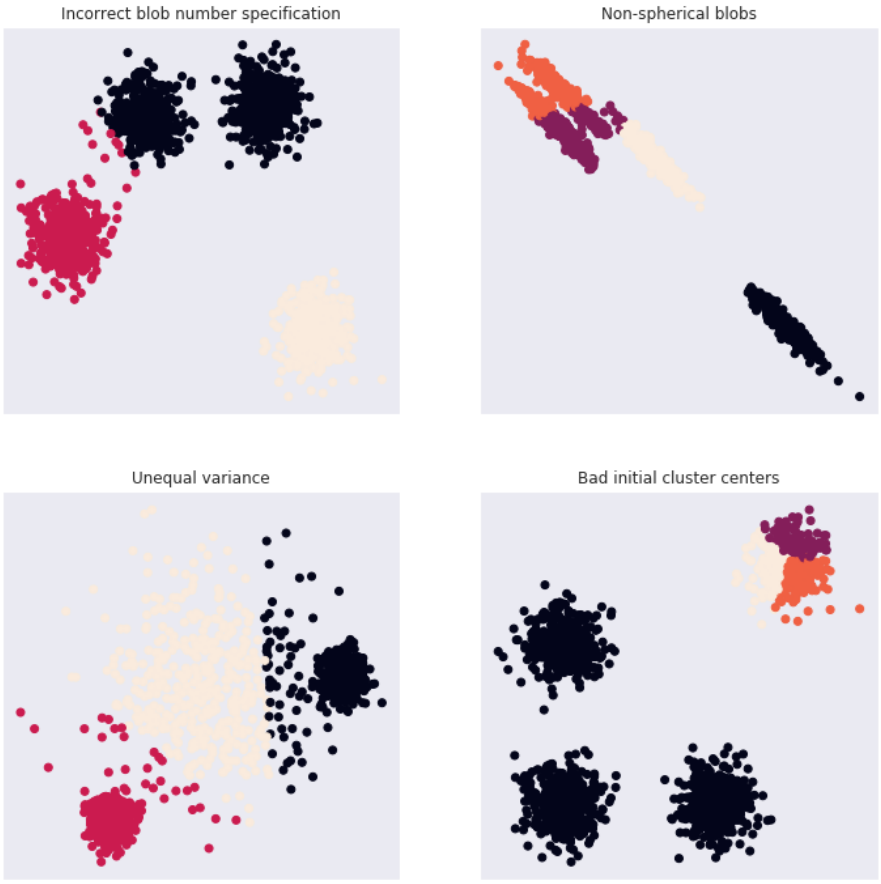


Figure 29.4.2: K-means performance can be affected by a number of factors, including incorrect number of clusters/blobs, non-spherical clusters/blobs, clusters/blobs with unequal variance, and bad initial cluster centers.

29.4.2.2 *K means++*

As we showed in subsection 29.4.2, the choices of initial cluster center can strongly affect the final results. K-means++ is an algorithm for choosing the initial cluster centers for the K-means clustering algorithm in order to avoid possible poor clustering found by the standard k-means algorithm.

The intuition behind this approach is that spreading out the K initial cluster centers can in general improve the final result. In K-means++, the first cluster center is chosen uniformly at random from the data points that are being clustered, after which each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the point's closest existing cluster center.

We summarize the initial seed selection into following steps:

Methodology 29.4.1 (K mean++ initial seeds selection procedure).
1. *Choose one center c_1 uniformly at random from among the data points X.*
2. *For each data point $x \in X$, compute $D(x) = \min_{c_i \in C} \|x - c_i\|$*
3. *Choose one new center $c_i \in X - C$ with probability $p \propto D^2(x)$. $C = C \cup c_i$*
4. *Repeat Steps 2 and 3 to get all k centers.*

A comparison of K-means and K mean++ algorithm is showed in Figure 29.4.3.

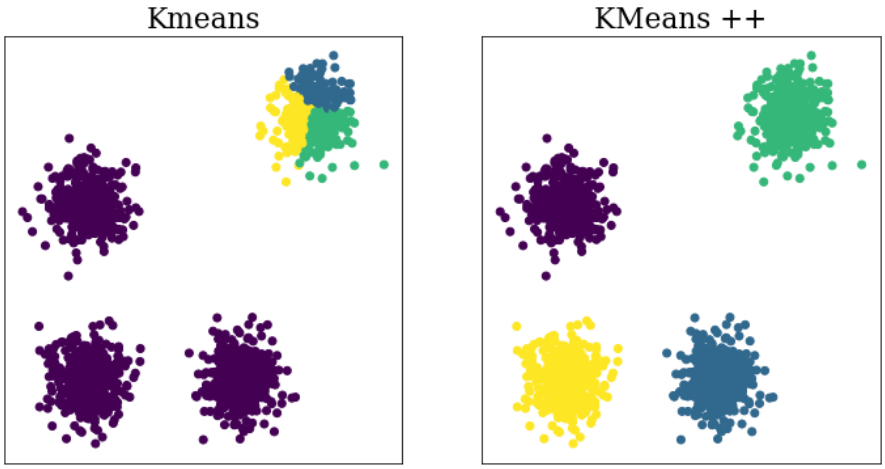


Figure 29.4.3: Clustering comparison between Kmeans and Kmeans++.

29.4.2.3 *Kernel K means*

The standard K means algorithm uses Euclidean distance to identify neighborhood and proximity between points. For some applications, such metric is not the best measure of distance. A simple remedy is using the Kernel trick [section 22.6], and the resulting algorithm is called Kernel K means.

One special case of Kernel K means algorithm is spectral clustering, which will be covered in subsection 29.4.4

29.4.3 Density-based spatial clustering of applications with noise (DBSCAN)

Density-based spatial clustering of applications with noise, or DBSCAN is another influential data clustering algorithm[26]. The core idea is to view the data points as a collections of graph nodes. Graph nodes are connected by edges if they are close enough to each other. Roughly, connected graph components are the clusters we look for.

To facilitate the introduction of the algorithm, here we define the key concepts of core points, reachable points and outliers:

- A point p is a **core point** if at least $minPts$ points are within distance ϵ (including itself).
- A point q is reachable from p if point q and p are in the same component of ϵ graph. An ϵ graph is a graph constructed on points and graph edges are created for nodes whose pairwise distances are smaller than ϵ.
- All points not reachable from any other point are outliers or noise points.
- The graph components constructed from the core points and their reachable points are the clusters.

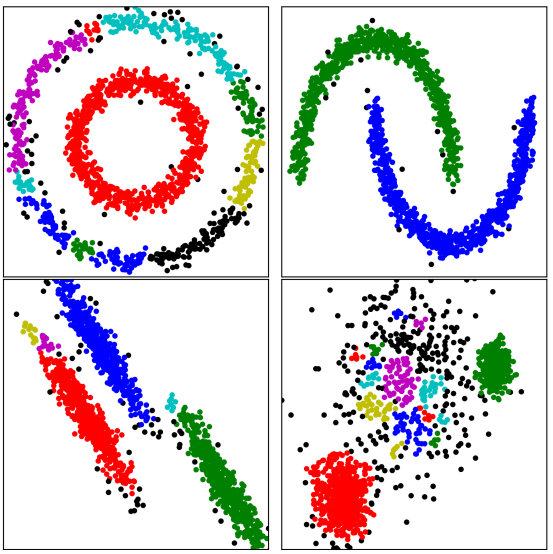
The procedure of DBSCAN is given in algorithm 59. The parameter $minPts$ is usually chosen based on the dimensionality of the data point, for example, $minPts \geq 2D$. After choosing $minPts$, the value ϵ can then be chosen by using a k-distance plot, a plot of the $k(k = minPts - 1)$ nearest-neighbor distances, computed for each point, and plotted from largest to smallest value. A good choice of ϵ is at where this plot shows an elbow[27].

Algorithm 59: DBSCAN algorithm

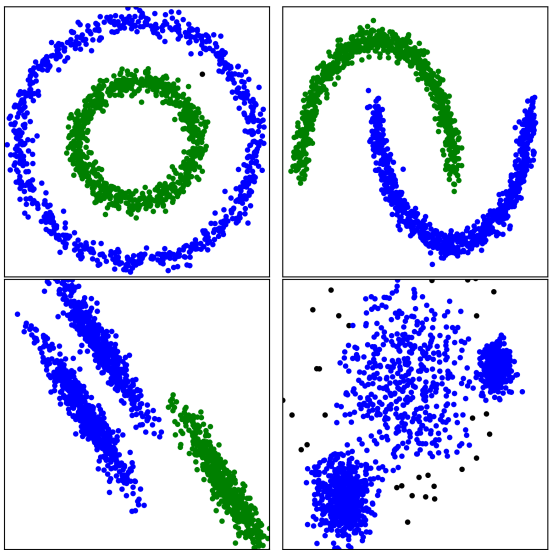
Input: data set $X = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, threshold ϵ, MinPts.

```
repeat
    all points are processed.
until Retrieve a point x ∈ X that has not been processed.
if x is core point then
    x and all the unprocessed reachable points in X form a new cluster C
end
;
```

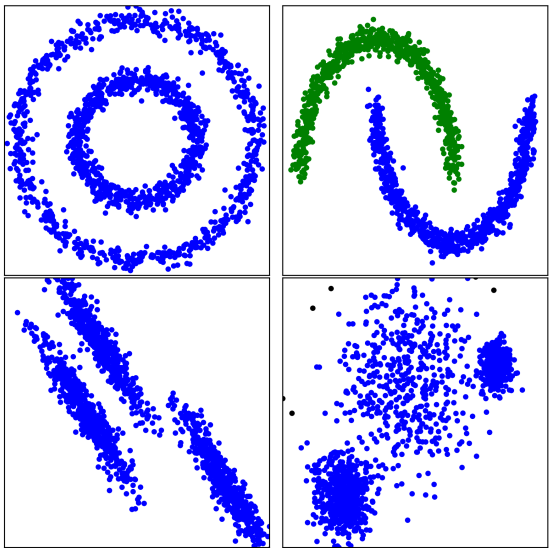
Output: the clustering result.



(a) $\epsilon = 0.1$.



(b) $\epsilon = 0.2$.



(c) $\epsilon = 0.4$.

Figure 29.4.4: DBSCAN demo with different ϵ.

DBSCAN algorithm has a number of strengths:

- Unlike K-means algorithm, DBSCAN does not require the user to specify the number of clusters in the data, as opposed to k-means.
- DBSCAN can deal with arbitrarily shaped clusters, whereas K-means algorithm can only handle spherical-shaped data.

And its weaknesses are listed as below

- The parameters $minPts$ and ϵ has to be specified by the user. Usually it is a non-trivial task for high-dimensional data.
- DBSCAN only employs a set of $minPts$ and ϵ to construct the grpah, therefore, it cannot cluster data sets with large differences in densities.
- The quality of DBSCAN highly hinge on the distance measure. The most common distance metric used is Euclidean distance, but euclidean distance is inadequate for high-dimensional data and other applications involving text, image, and time series.

The strengths and weaknesses can be illustrated by the results in Figure 29.4.4.

29.4.4 Spectral clustering

The core idea of spectral clustering is to use Laplacian eigenmap to find a suitable low dimensional representation that maximally preserve some distance measure between data points, and then use K-means to perform clustering based on this distance measure.

To perform Laplacian eigenmap, we need to first create a neighborhood graph similar to DBSCAN. However, different from DBSCAN that directly performs clustering on the graph, spectral clustering seeks clustering based on the low dimensional representation derived the spectrum of the Laplacian of the graph.

We can review the definition and basic properties of graph Laplacian are covered in subsection 29.3.5. The algorithm is presented below.

Algorithm 60: General spectral clustering algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$

1 Construct a similarity graph. Let W be the weighted adjacency matrix.
2 Compute a Laplacian L or (L_{sys}, L_{rw}).
3 Compute the first top k eigenvectors $u_1, .., u_k \in \mathbb{R}^N$ of L.
4 Assign each data point i with the new lower dimensional coordinate $y_i = (u_{1,i}, u_{2,i}, ..., u_{k,i})$.
5 Use K-means algorithms to do clustering on the lower dimensional coordinates.

Output: clustered results

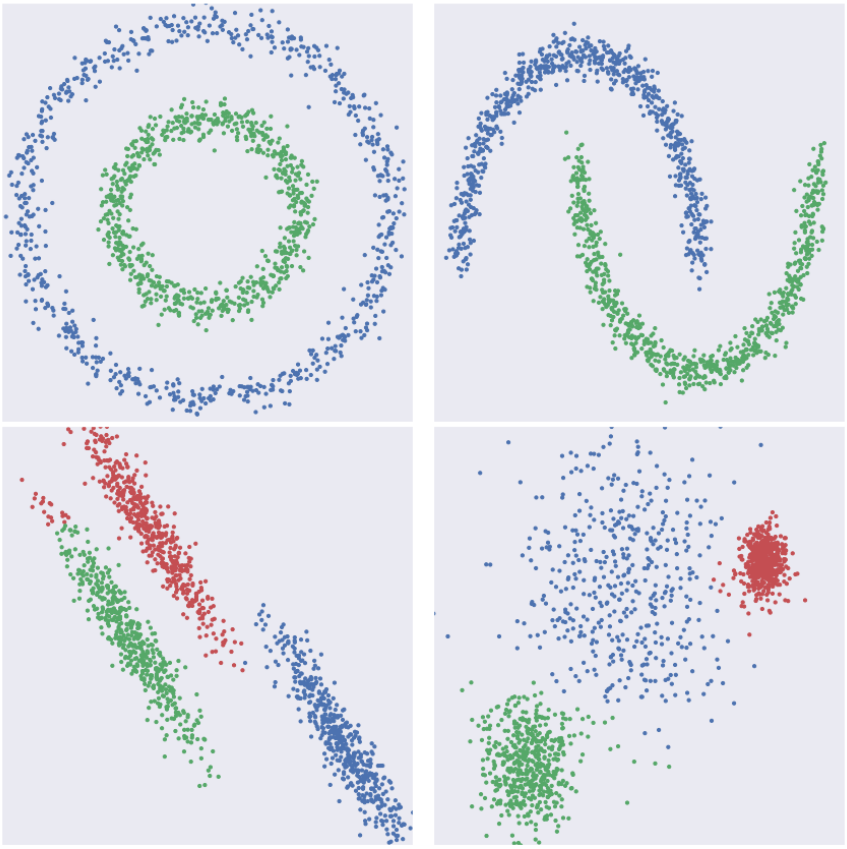


Figure 29.4.5: Spectral clustering demo.

Remark 29.4.1.

- Spectrum clustering is more computational expensive than DBSCAN.
- The spectrum of the Laplacian provides additional information on the number of components. In fact, the number of zero eigenvalues is the number of connected component in the graph. We can use this as the benchmark to understand final cluster results.

29.4.5 Gaussian mixture models (GMM)

29.4.5.1 *Preliminaries: Expectation Maximization (EM) algorithm*

Consider a statistical model with model parameter θ that generate a set of observed data $X = \{x_1, ..., x_n\}$ and a set of unobserved latent data or missing data $Z = \{z_1, ..., z_N\}$. Denote the likelihood function for X, Z by $L(\theta; X, Z)$ and the generation probability model

by $p(X, Z|\theta)$. The MLE for θ is given by maximizing the marginal likelihood of X, given by

$$L(\theta; X) = p(X|\theta) = \int p(X, Z|\theta) dZ.$$

Example 29.4.1 (Hidden Markov chain). Consider the graphical model (E) representing a hidden Markov chain. The joint probability can be decomposed by

$$\begin{aligned} &P(X_1, ..., X_N, Z_1, ..., Z_N) \\ &= P(Z_N|Z_{N-1})P(Z_{N-1}|Z_{N-2}) \cdots P(Z_1) \prod_{n=1}^{N} P(X_i|Z_i). \end{aligned}$$

Besides trying to learn the hidden state observation $z_1, ..., z_N$ from $x_1, ..., x_N$, the model coefficient θ to be learned includes transition model coefficient characterizing $P(Z_i|Z_{i-1})$ and emission model coefficient $P(X_i|Z_i)$.

Optimizing $L(\theta; X)$ is usually challenging because integrating Z can be computational intractable. The EM algorithm is an efficient iterative procedure to compute the MLE in the presence of missing or hidden data. The EM algorithm proposes a two-step iterative procedure to solve the optimization problem.

- Expectation step (E step). Compute averaged $L(\theta; X, Z)$ by over the conditional distribution of Z given X. That is, compute

$$Q(\theta|\theta^{(t)} = E_{Z\sim Z|X,\theta^{(t)}}[\log L(\theta; X, Z)]$$

- Maximization step (M step). Solve optimization

$$\theta^{(t+1)} = \arg\max_{\theta} Q(\theta|\theta^{(t)}).$$

Now we are going to show that the two steps in the EM algorithm can guarantee the improvement of the log likelihood function $\log L(\theta; X)$. The essential strategy is to show that

$$Q(\theta^{(t+1)}|\theta^{(t)}) \geq Q(\theta^{(t)}|\theta^{(t)} \implies \log p(X|\theta^{(t+1)}) \geq \log p\left(X|\theta^{(t)}\right).$$

Theorem 29.4.1 (improvement of likelihood in EM steps). *At each iteration t, the increase of $\log p(X|\theta)$ from $\log p\left(X|\theta^{(l)}\right)$ has a lower bound given by $Q(\theta|\theta^{(t)} - Q(\theta^{(t)}|\theta^{(t)}$*

Since in the M step, we have $Q(\theta^{(t+1)}|\theta^{(t)} \geq Q(\theta^{(t)}|\theta^{(t)}$, then the EM algorithm guarantees

$$\log p(X|\theta^{(t+1)}) \geq \log p\left(X|\theta^{(t)}\right).$$

Proof. Note that based on conditional probability we can write

$$\log p(X|\theta) = \log p(X, Z|\theta) - \log p(Z|X, \theta)$$

We can multiply terms on the right by $\sum_Z p\left(Z|X, \theta^{(t)}\right)$, which has value 1, and get

$$\begin{aligned}\log p(X|\theta) &= \sum_Z p\left(Z|X, \theta^{(t)}\right) \log p(X, Z|\theta) - \sum_Z p\left(Z|X, \theta^{(t)}\right) \log p(Z|X, \theta) \\ &= Q\left(\theta|\theta^{(t)}\right) + H\left(\theta|\theta^{(t)}\right)\end{aligned}$$

where $H(\theta|\theta^{(t)}) = -\sum_Z p(Z|X, \theta^{(t)}) \log p(Z|X, \theta)$.

Subtracting the equation at θ and $\theta^{(t)}$, we have

$$\log p(X|\theta) - \log p\left(X|\theta^{(t)}\right) = Q\left(\theta|\theta^{(t)}\right) - Q\left(\theta^{(t)}|\theta^{(t)}\right) + H\left(\theta|\theta^{(t)}\right) - H\left(\theta^{(t)}|\theta^{(t)}\right).$$

From the non-negativeness of KL divergence [Lemma 11.14.2], we have

$$H\left(\theta|\theta^{(t)}\right) \geq H\left(\theta^{(t)}|\theta^{(t)}\right),$$

so we can conclude that

$$\log p(X|\theta) - \log p\left(X|\theta^{(t)}\right) \geq Q\left(\theta|\theta^{(t)}\right) - Q\left(\theta^{(t)}|\theta^{(t)}\right).$$

□

29.4.5.2 *The GMM model and algorithm*

Given a set of points $x_1, ..., x_T \in \mathbb{R}^D$, the goal is to find a data generation model

$$p(x|\theta) = \sum_{i=1}^{M} w_i g_i(x|\mu_i, \Sigma_i),$$

where the mixture weights satisfying $\sum_{i=1}^{M} w_i = 1$, parameterized by $\theta = \{w_i, \mu_i, \Sigma_i, i = 1, ..., M\}$, given as

$$g(x|\mu_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp(-(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i))$$

such that likelihood for the observed data

$$L = \prod_{i=1}^{T} p(x_i|\theta)$$

is maximized.

Let $z = (z_1, ..., z_n)$ be the latent variables that determine the component from which the observation originates.

The EM algorithm for GMM consists of the following two iterative steps:

- E step: for given parameter values we can compute the distribution of z given X based on Bayesian rule.
- Maximization step: updates the parameters of our model based on the latent variable calculated using ML method.

Algorithm 61: Gaussian mixture model EM algorithm

Input: Data set consists of $x_1, x_2, ..., x_N$

1 Set $n = 0$, and set initial values θ.

2 E step:

$$Pr(i|x_t, \theta) = \frac{w_i g(x_t|\mu_t, \Sigma_t)}{\sum_{k=1}^{M} w_k g_i(x_k|\mu_k, \Sigma_k)}, \forall i \in \{1, ..., M\}, \forall t \in \{1, ..., T\}$$

3 M step: update θ as

4

$$w_i = \frac{1}{T} \sum_{i=t}^{T} Pr(i|x_t, \lambda), \forall i$$

5

$$\mu_i = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_i}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)}, \forall i$$

6

$$\sigma_i^2 = \frac{\sum_{t=1}^{T} Pr(i|x_t, \lambda) x_t^2}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} - \mu_i^2, \forall i$$

Output: the optimalized value θ.

Remark 29.4.2 (comparison with K-means). As we mentioned in subsection 29.4.2, the vanilla K-means in general cannot work with data with anisotropic shape or uneven variances. GMM overcomes the weakness by modeling the covariance structure for each cluster. In Figure 29.4.6, we can see that GMM can handle data with anisotropic or uneven variances.

We can regularize GMM by restricting the covariance structure in different ways, including enforcing diagonal covariance matrix, or constant covariance matrix.

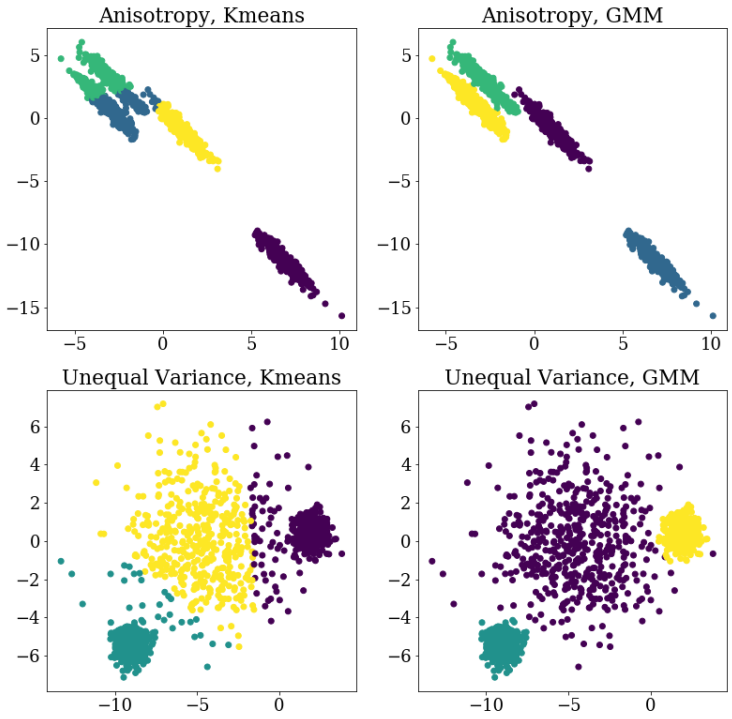


Figure 29.4.6: Clustering comparison between K-means and GMM.

29.4.6 Application examples

29.4.6.1 *Image segmentation*

Here we consider an application of K-means clustering algorithm for image segmentation, which aims to group visually similar parts together. In Figure 29.4.7, we seek K clusters based on each pixel's RGB value and then substitute each pixel's RGB value by their assigned cluster's mean value. The K value specifies the number of groups the image will be segmented to. At large K value (e.g., K=64), the clustering procedure is also called color quantization.

Here we chosen RGB value as the feature for each pixel; there are other options on the features, including pixel's position that encodes proximity information.

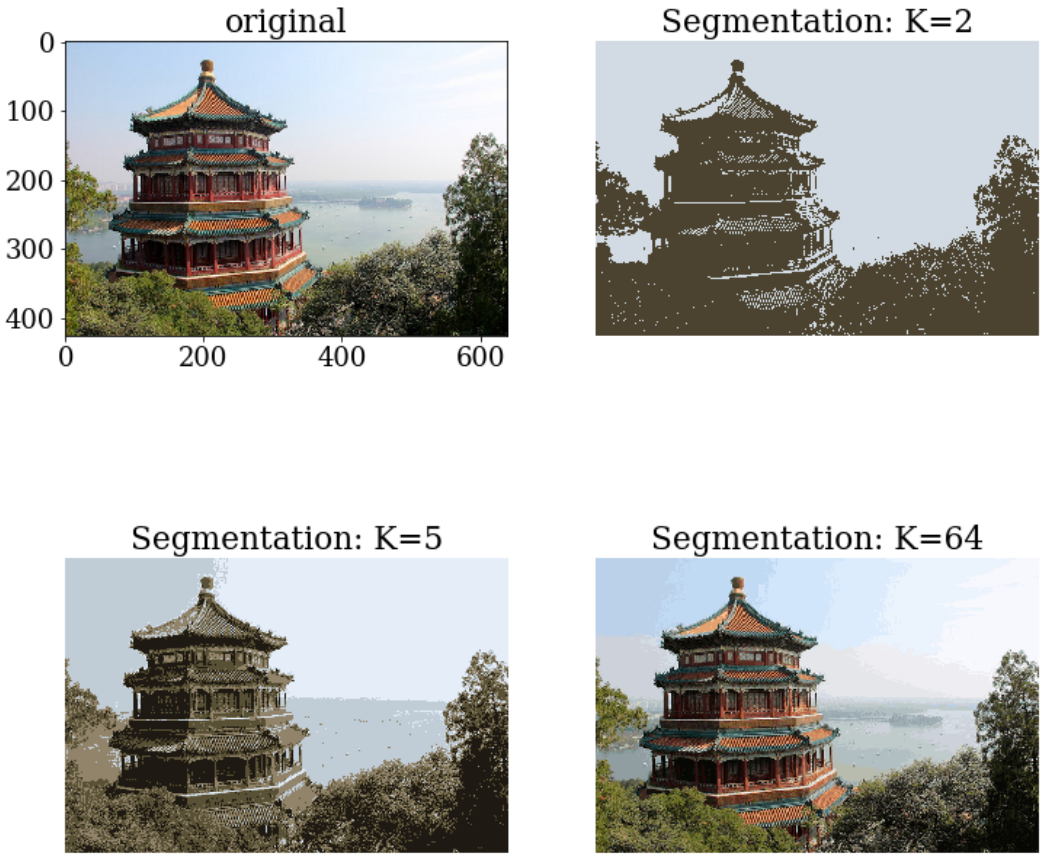


Figure 29.4.7: K-means application to image segmentation.

29.5 Notes on Bibliography

Excellent treatment on PCA and its generalization, see [3].

For machine learning theory, see [28]. For multidimensional scaling, see[29]. For spectral clustering, see [21]. For recommendation systems, see [8].

BIBLIOGRAPHY

1. Ma, Y. & Vidal, R. Generalized principal component analysis. *Unpublished Notes* (2002).
2. Candès, E. J., Li, X., Ma, Y. & Wright, J. Robust principal component analysis? *Journal of the ACM (JACM)* **58,** 1–37 (2011).
3. Vidal, R., Ma, Y. & Sastry, S. S. in, 63–122 (Springer, 2016).
4. Cai, T. T. & Wang, L. *Orthogonal matching pursuit for sparse signal recovery with noise* in (2011).
5. Mairal, J., Bach, F., Ponce, J. & Sapiro, G. *Online dictionary learning for sparse coding* in *Proceedings of the 26th annual international conference on machine learning* (2009), 689–696.
6. Lee, D. D. & Seung, H. S. Learning the parts of objects by non-negative matrix factorization. *Nature* **401,** 788–791 (1999).
7. Lee, D. D. & Seung, H. S. *Algorithms for non-negative matrix factorization* in *Advances in neural information processing systems* (2001), 556–562.
8. Aggarwal, C. C. *et al. Recommender systems* (Springer, 2016).
9. Koren, Y., Bell, R. & Volinsky, C. Matrix factorization techniques for recommender systems. *Computer,* 30–37 (2009).
10. Peña, D. & Poncela, P. in *Advances in distribution theory, order statistics, and inference* 433–458 (Springer, 2006).
11. Linden, G., Smith, B. & York, J. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing,* 76–80. ISSN: 1089-7801 (2003).
12. Adomavicius, G. & Tuzhilin, A. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions* 2005.
13. Baker, D. A surprising simplicity to protein folding. *Nature* **405,** 39. ISSN: 1476-4687 (2000).
14. Onuchic, J. N., Luthey-Schulten, Z. & Wolynes, P. G. Theory of protein folding: the energy landscape perspective. *Annual review of physical chemistry* **48,** 545–600. ISSN: 0066-426X (1997).
15. Perlmutter, J. D. & Hagan, M. F. Mechanisms of virus assembly. *Annual review of physical chemistry* **66,** 217–239. ISSN: 0066-426X (2015).

16. Tang, X. *et al.* Optimal Feedback Controlled Assembly of Perfect Crystals. *ACS nano* **10,** 6791–6798 (2016).

17. Edwards, T. D., Yang, Y., Beltran-Villegas, D. J. & Bevan, M. A. Colloidal crystal grain boundary formation and motion. *Scientific Reports* **4,** 6132 (2014).

18. Yang, Y., Thyagarajan, R., Ford, D. M. & Bevan, M. A. Dynamic colloidal assembly pathways via low dimensional models. *The Journal of chemical physics* **144,** 204904 (2016).

19. Bengio, Y. *et al.* Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in Neural Information Processing Systems,* 177–184. ISSN: 10495258 (2003).

20. Tenenbaum, J. B., De Silva, V. & Langford, J. C. A global geometric framework for nonlinear dimensionality reduction. *science* **290,** 2319–2323 (2000).

21. Von Luxburg, U. A tutorial on spectral clustering. *Statistics and computing* **17,** 395–416 (2007).

22. De la Porte, J, Herbst, B., Hereman, W & Van Der Walt, S. *An introduction to diffusion maps* in *The 19th Symposium of the Pattern Recognition Association of South Africa* (2008).

23. Miranda, H.-C. *Applied Stochastic Analysis lecture notes* (NewYork University, 2015).

24. Bengio, Y. *et al.* Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. *Advances in neural information processing systems* **16,** 177–184 (2004).

25. Laing, C. R., Frewen, T. A. & Kevrekidis, I. G. Coarse-grained dynamics of an activity bump in a neural field model. *Nonlinearity* **20,** 2127 (2007).

26. Ester, M., Kriegel, H.-P., Sander, J., Xu, X., *et al. A density-based algorithm for discovering clusters in large spatial databases with noise.* in *Kdd* **96** (1996), 226–231.

27. Schubert, E., Sander, J., Ester, M., Kriegel, H. P. & Xu, X. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)* **42,** 1–21 (2017).

28. Mitchell, T. M. *et al. Machine learning. WCB* 1997.

29. Borg, I. & Groenen, P. J. *Modern multidimensional scaling: Theory and applications* (Springer Science & Business Media, 2005).